VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

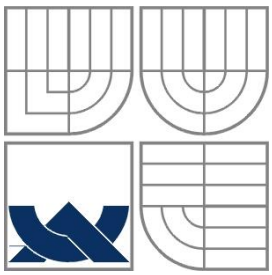
APLIKACE PRO KOMUNIKACI SE ZTRACENÝM MOBILNÍM TELEFONEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

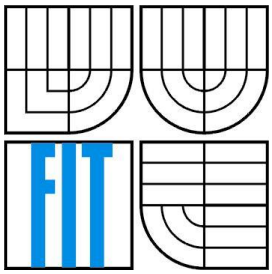
AUTOR PRÁCE
AUTHOR

Bc. PETR SLÁDEK

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

APLIKACE PRO KOMUNIKACI SE ZTRACENÝM MOBILNÍM TELEFONEM

APPLICATION FOR COMMUNICATION WITH THE LOST MOBILE PHONE

DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR SLÁDEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PAVEL OČENÁŠEK, Ph.D.

BRNO 2015

Abstrakt

Tato práce se věnuje problematice komunikace ztraceného mobilního zařízení pomocí internetu s jeho majitelem kvůli možnostem nalezení, navrácení či zablokování zařízení. Zaměřuje se na analýzu existujících řešení a návrh vlastní mobilní aplikace pro platformu Android s podpůrnou webovou aplikací. Práce také shrnuje základní principy tvorby aplikací pro Android OS a komunikaci přes cloudovou službu Google Cloud Messaging.

Abstract

This thesis is dedicated to communication with lost mobile device throw the Internet with its owner because of the possibility of finding, return or lock the device. It focused on analysis existing solution and draft own mobile application for Android platform with supportive web application, This thesis also summary basic principles of creating application for Android OS and communication with cloud base service Google Cloud Messaging.

Klíčová slova

Android, Java, Google Cloud Messaging, GCM, Ztracený telefon, PHP, Java

Keywords

Android, Java, Google Cloud Messaging, GCM, Lost Phone, PHP, Java

Citace

Sládek Petr: Aplikace pro komunikaci se ztraceným mobilním telefonem, diplomová práce, Brno, FIT VUT v Brně, 2015

Aplikace pro komunikaci se ztraceným mobilním telefonem

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Pavla Očenáška, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Bc. Petr Sládek
28. května 2015

Poděkování

Chtěl bych poděkovat svému vedoucímu práce Ing. Pavlu Očenáškovi, Ph.D. za ochotu při vedení práce, poskytnutí připomínek, konzultací a odborných rad. Dále bych rád poděkoval všem, kteří mě při psaní této práce jakkoli podporovali.

Obsah

1 Úvod

Mnou vybrané téma jsem zvolil především z důvodu, že operační systém Android stále drží přední příčku na trhu, že jeho nasazování se rozšiřuje na čím dál tím více druhů zařízení. Proto jistě stojí za to zdokonalit se co nejvíce ve vývoji software právě pro tuto platformu. Téma diplomová práce zahrnuje velmi širokou oblast využití všech služeb, které platforma nabízí.

Návrh a implementaci aplikace pro komunikaci se ztraceným mobilním telefonem jsem zvolil proto, že nativní aplikace od Googlu k tomuto účelu nemá mnoho funkcí k nalezení telefonu v případě ztráty či odcizení. Cílem této práce bude navrhnout a implementovat funkce, které usnadní případnému nálezci telefonu snadno zkontaktovat jeho majitele či funkce ke snadnému nalezení telefonu v případě jeho ztráty třeba jen v bytě či autě. Samozřejmě také bude počítat ze situací, kdy telefon někdo nalezne, ale nebude ho chtít vrátit. To vše pomocí aplikace pro mobilní zařízení a webové aplikace v prohlížeči, které spolu budou komunikovat přes internet.

Tato práce se zabývá analýzou vývoje pro operační systém Android, komunikací uživatele s mobilním zařízením pomocí internetu přes službu Google Cloud Messaging a srovnáním konkurenčních řešení. Dále pak specifikací požadavků na aplikaci jejich návrh, implementaci a testování.

2 Platforma Android

Android je softwarová platforma určená pro mobilní zařízení. V dnešní době jsou to především chytré mobilní telefony a tablety, ale na vzestupu jsou už i chytré hodinky, televize či automobily. Vývoj Androidu obstarává asociace Open Handset Alliance, jejíž členy jsou mobilní operátoři, výrobci mobilních telefonů, softwarové firmy a další. Oproti dvěma svým největším konkurentům iOS a Windows Phone, je Android dostupný jako otevřená platforma (Open Source). [1][4]

Platforma Android zahrnuje operační systém (založený na Linuxovém jádře), softwarové prostředí (tzv. middleware), uživatelské rozhraní a aplikace. Při vývoji je kladen důraz na omezení mobilních zařízení a proto je optimalizován pro menší spotřebu baterie, či pro nižší hardwarové požadavky. [1][2]

2.1 Historie

Existence Androidu se vyvíjí od roku 2003. V tomto roce vznikla v Palo Alto v USA ve státě Kalifornie společnost s názvem Android, Inc. V čele společnosti stáli Andy Rubin, Rich Miner, Nick Sears (víceprezident v T-mobile) a Chris White. Původně bylo jejich myšlenkou vytvořit operační systém který by se dal použít pro digitální fotoaparáty. Chvíli po té co začali s vývojem zvážili znovu budoucnost, a protože trh s digitálními fotoaparáty v té době nebyl tak velký a rozsáhlý zaměřili se na chytré mobilní telefony. [7]

V srpnu roku 2005 nastal klíčový moment. Tehdy téměř neznámou společnost Android, Inc koupil v tichosti gigant Google a udělal z ní svoji 100% dceřinou společnost. Na vývoji se stále podíleli její spoluzakladatelé a stále platilo to, že Android nebyl nijak známí. Nicméně veřejnost začala tušit, že Google má s Androidem velké plány a že bude chtít vstoupit na trh s mobilními telefony, což se také brzy ukázalo jako pravda. [7]

Společnost Google, prostřednictvím týmu vedeného Andy Rubinem, vyvinula mobilní platformu s linuxovým jádrem a začala ji prosazovat na mobilní trh. Výrobcům telefonů i mobilní operátorům přesvědčili, že poskytují flexibilní a hlavně inovativní systém. Uzavřeli celou řadu partnerství kolem hardwarových i softwarových komponentů na různých úrovních. V této době Google získal několik patentů v oblasti mobilních technologií. [7]

Oficiálního představení inovativního mobilního operačního systému Android, se veřejnost dočkala 5.listopadu 2007. Vývoj jeho první verze tedy trval čtyři roky, a to ještě před odkoupením Googlem několikrát hrozilo, že společnost zbankrotuje. Android byl první software pro mobilní telefony, který byl postaven na linuxovém jádru ve verzi 2.6. K tomuto datu se také váže vytvoření uskupení Open Handset Alliance. [7]

Až za rok po představení Androidu se na trhu objevil první chytrý telefon, který měl tento operační systém. Byl jím HTC Dream s hardwarovou klávesnicí (známí také jako T-mobil G1) a vyšel 22.října 2008. Na český trh se dostal až na počátku roku 2009. V této době byla také zprovozněná veřejná beta verze Android Marketu (dnes se jmenuje Google Play Store), na kterém bylo tehdy cca 50 aplikací (dnes je to něco přes milion). [7]

Verze OS	Kódové označení	API	Rozšíření
2.2	Froyo	8	0.6%
2.3.3 – 2.3.7	Gingerbread	10	9.8%
4.0.3 – 4.0.4	Ice Cream Sandwich	15	8.5%
4.1.x	Jelly Bean	16	22.8%
4.2.x		17	20.8%
4.3		18	7.3%
4.4	KitKat	19	30.2%

Tabulka 1 Přehled verzí a procentuální rozšíření z 3.11.2014; Zdroj [8]; Provedení: Vlastní

2.2 Současná situace

Nyní, koncem roku 2014, přichází na trh nová verze s názvem Android 5.0 Lollipop. Zde se jako v každé nové verzi objevilo hned několik zajímavých novinek. Asi nejdiskutovanější z nich je tzv. „Material Design“. Ten přináší do androidu nové grafické UI, které má intuitivněji reagovat na pohyb a lépe využívat celou plochu displeje. Také přináší nové pastelové barvy, novou typografii, lepší stíny a osvětlení. Tyto změny mají přispět k tomu, aby prostředí OS android vypadalo napříč všemi zařízeními (hodinky, telefony, tablety, televize,..) stejně a na všech těchto zařízeních se s ním dobře pracovalo. [12]

Nová verze taky přináší lepší správce notifikací, který má možnost notifikace filtrovat a nebo naopak některé zobrazit i na lockscreen[1]. Dalším užitečnou novinkou je lepší správa napájení. Díky ní může zařízení vydržet na baterii až od 90 minut déle. Lze také zjistit jak dlouho ještě musí být telefon na nabíječce než se plně nabije a nebo čas jak dlouho ještě na baterii telefon vydrží než se úplně vybije.

Dalšími novinkami jsou například uživatelské profily či profil hosta, které mají přístup jen ke svým datům, a aplikacím které jim povolíte. Dále jsou zde novinky v oblasti zabezpečení, komunikace, rychlosti atd. [12]

[1] Obrazovka zařízení po stisknutí vypínacího tlačítka, kdy je zařízení zamknuté.

Aby bylo vidět, jaký podíl na trhu dnes jednotlivé platformy zastávají, uvedu zde pár čísel. V polovině roku 2014 obsadil Android téměř 85% trhu s mobilními zařízeními. Až daleko za ním následuje iOS s 11,7% a Windows Phone jen s 2.5%. Ostatní platformy a předešlé roky lze vidět na následující tabulce.

Kvartál	Android	iOS	Windows Phone	BlackBerry OS	Ostatní
Q2 2014	84.7%	11.7%	2.5%	0.5%	0.7%
Q2 2013	79.6%	13.0%	3.4%	2.8%	1.2%
Q2 2012	69.3%	16.6%	3.1%	4.9%	6.1%
Q2 2011	36.1%	18.3%	1.2%	13.6%	30.8%

Tabulka 2 Podíl na trhu mobilních telefonů; Zdroj [5]; Provedení: Vlastní

2.3 Architektura

Android má svou architekturu rozdělenou na 5 klíčových vrstev. Na obrázku Obrázek 1 jsou schematicky zakreslené. Vezmeme je odzdola nahoru a rozebereme jejich účel.

Obrázek 1 Vrstvy architektury; Zdroj [6]; provedení vlastní

2.3.1 Linuxové jádro

Nejnižší vrstvou architektury je linuxové jádro operačního systému. Tvoří abstrakci mezi hardwarem konkrétního mobilního zařízení a softwarem vyšších vrstev. Je postaveno na Linuxu ve verzi 2.6 (později na 3.x). Obstarává například podporu správy paměti, správy procesů, správy sítí a ovladačů hardware. Oproti běžným linuxovým distribucím, zde ale nenajdeme podporu grafického rozhraní X Window Systém ani nemá úplnou sadu GNU knihoven, které nejsou pro mobilní zařízení potřeba. Linuxové jádro má také tu výhodu, že je snadno sestavitelné na různých zařízeních a tím přenositelné. [3]

2.3.2 Knihovny a Android Runtime

Další vrstva po jádru jsou knihovny. Jsou napsané v programovacím jazyce C nebo C++. Řadí se do nich například knihovny pro Media (pro přehrávání různých audio a video formátů), OpenGL knihovna pro 3D grafiku, knihovna LibWebCore pro vykreslování HTML obsahu webových stránek, dále také knihovna pro SQLite uložiště, knihovny Apache pro práci ze sítí nebo knihovna pro SSL. A samozřejmě mnohé další. [3]

Do této vrstvy patří taky tzv. Android Runtime obsahující Dalvik Virtual Machine, což je aplikační virtuální stroj, vyvíjen speciálně pro android v Googlu. Důvod pro vznik vlastního virtuálního stroje pro Javu, byl především ten, že běžný Java Virtual Machine od Oracle není volně šířitelný a také není optimalizovaný pro mobilní zařízení, co se úspory energie a výkonu týče. VM se používal až do verze 4.3, poté byl nahrazen zpětně kompatibilním Dalvik Turbo od francouzko švýcarské firmy Myriad Group. Dalvik Turbo je výrazně rychlejší a úspornější. [2][3]

Od Android verze 4.4 včetně, také probíhá jinak překlad aplikace. Do této verze se zdrojový kód v Javě přeložil do Java byte kódu pomocí standardního překladače a poté se ještě tento byte kód překompiloval pomocí Dalvik kompilátoru na výsledný Dalvik byte kód spustitelný v DVM. Nově se používá tzv. dopředná kompilace (AOT – Ahead-of-time compilation), která zajištuje větší úspory energie a výrazné zrychlení aplikací. Funguje to tak, že zdrojový kód aplikace v Javě se opět přeloží do Java byte kódu a poté do Dalvik byte kódu, který se ovšem hned nespouští, ale při instalaci aplikace se tento dalvik kód jednou provždy zkompiluje do nativního kódu procesoru daného zařízení. Instalace aplikace sice trvá o něco déle, kvůli finální optimalizaci, ale výdrž telefonu je až o 36% větší a výkon zařízení až dvojnásobný [2][9].

2.3.3 Aplikační Framework

Předposlední vrstva je důležitá pro vývojáře aplikací. Nabízí přístup k velkému množství služeb potřebných pro tvorbu aplikace. Jsou to například prvky uživatelského rozhraní, notifikace, služby na pozadí, poskytování nebo čtení obsahu mezi aplikacemi, spouštění jednotlivých obrazovek, nebo jiných aplikací a tak podobně. Aplikačním frameworkem se budu zabývat podrobněji v další kapitole 2.4. [2]

2.3.4 Samotné aplikace

Poslední a nejdůležitější vrstvou z pohledu uživatele jsou samotné aplikace. Patří k nim základní aplikace jako Telefon, SMS zprávy, Kontakty, Kalendář, Fotoaparát apod. které jsou součástí Operačního systému a nelze je odinstalovat. Dále pak další „nepovinné“ aplikace od Googlu i od dalších developerů. Sem spadají všechny ostatní aplikace jako např. Hry, Webové prohlížeče, různé aplikace pro správu či aplikace kin, taxi služeb, restaurací a podobně. Do této vrstvy spadá i naše aplikace pro komunikaci se ztraceným telefonem. [2]

2.4 Stavební prvky aplikace

Aplikační Framework operačního systému android nabízí rozhraní (služby) ze kterých se skládají všechny uživatelské aplikace. Samozřejmě ne každá aplikace musí využívat vše, co systém nabízí. Jednotlivé prvky, ze kterých se aplikace staví, jsou Activity, Views, Resource, Intent, Service, Broadcast reciver a Content Provider. Jednotlivé pojmy jsou popsány v následujících podkapitolách.

2.4.1 Activity

Aktivitu si můžeme představit jako jednu obrazovku aplikace. Plní funkci podobnou například Controlleru v MVC aplikacích. Slouží k tomu, aby data získaná z jiných vrstev aplikace (Databáze, REST API, Content Provideru,…) správně poskytla uživateli prostřednictvím View (Uživatelského rozhraní) a také obsloužila uživatelské vstupy (akce po stisknutí tlačítka, po přejetí prstem přes displej apod.). Každá Activity v aplikaci je potomkem třídy `android.app.Activity` a v průběhu práce s operačním systémem může přicházet do různých stavů. [1][13]

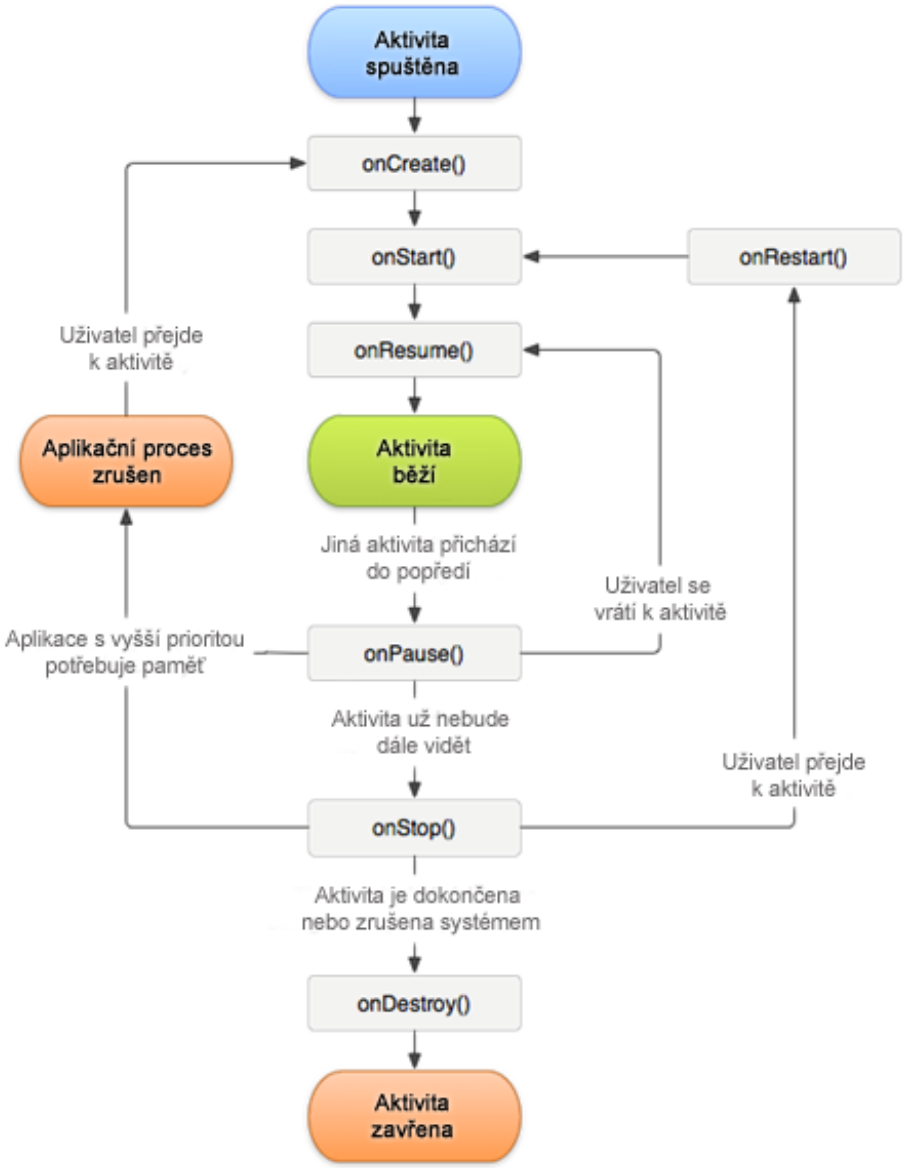
Životní cyklus Activity je vidět na obrázku Obrázek **2.** Aplikace může být buď spuštěná (zatím pozastavená v pozadí), běžící v popředí nebo ukončená.

2.4.2 View

View, neboli pohled, tvoří uživatelské rozhraní. Jsou to jednak různé formulářové prvky, textové položky, obrázky, seznamy položek a další, ale také tzv. layouty. Například `LinearLayout` řadí všechny prvky v sobě pod sebe (nebo vedle sebe), `RelativeLayout` dává všechny prvky přes sebe, `GridLayout` do mřížky atd.

Každá aktivita si své View vytvoří a to buď ze zadaného XML souboru definující strukturu pohledu (ten se dá tvořit i pomocí WYSIWYG v IDE) a nebo dynamicky, vytvářením jednotlivých

objektů a zanořováním jich do stromové struktury pohledu. V praxi se používá kombinace obou metod. Aktivita potom tyto prvky plní daty, mění je, maže je a podobně.



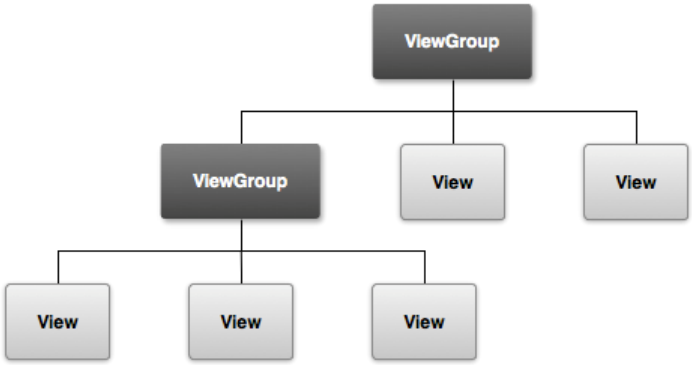
Obrázek 2 Životní cyklus Activity; Zdroj [13]; provedení vlastní

Prvky View jsou potomky třídy `android.view.View`. Layouty a seznamy a další, které mohou obsahovat další prvky v sobě, jsou potomky její podtřídy `android.view.ViewGroup`. Ty si lze představit jako rodičovské prvky v XML struktuře, nebo jako uzly stromu hierarchie View, jako je vidět na obrázku Obrázek **3**.

2.4.3 Resource

V případě, že potřebujete do aplikace přidat obrázek či zvuk přidáte je do speciální složky `/src/res/`, která obsahuje tzv. zdroje (resources). Tato složka může obsahovat následující podsložky:

- `drawable` - pro obrázky,
- `layout` - pro výše zmíněné XML soubory layoutů jednotlivých aktivit,
- `menu` - pro XML soubory definující kontextová menu,
- `raw` - pro obecné binární soubory,
- `values` - pro XML soubory s různými textovými řetězci, barvami, styly, čísly a podobně,
- `xml` – pro speciální XML soubory, které bude využívat vaše aplikace.



Obrázek 3 Hierarchie prvků uživatelského rozhraní; Zdroj [13]

Tyto zdroje se využívají, také pro překlady aplikace, nebo pro přízpůsobení designu pro různé rozlišení displejů. V souboru /res/values/strings.xml, jehož struktura je vidět na obrázku Obrázek **4** a to díky modifikátorům. Modifikátory jsou suffixy názvů dříve zmiňovaných složek, které slouží k výběru správné složky podle jazyku zařízení, dostupné výšky či šířky displeje, orientace zařízení (na výšku či vodorovně), hustoy pixelů nebo také verze API. Například složka /res/values-cs-land se použije pouze v případě české lokalizace a telefonu otočeného vodorovně. V jiném případě se použije jiná vyhovující složka, případně defaultní `/res/values/`.

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <!-- /res/values/strings.xml -->
3. <resources>
4.     <string name="firstname_label">Jméno:</string>
5.     <string name="lastname_label">Příjmení:</string>
6.     <string name="login">Přihlásit se</string>
7.     <string name="length_of_password">Krátké heslo.</string>
8. </resources>
```

Obrázek 4 Zdrojový kód XML souboru s textovými řetězci

2.4.4 Intent

Základním kamenem pro komunikaci mezi jednotlivými prvky aplikací jsou tzv. Intenty. Intent označuje jakýsi úmysl nebo záměr co chceme udělat a operační systém se postará o to aby spustil správnou aktivitu správné aplikace, která tento záměr umí vykonat, a vrátit do naší aplikace daná data. Například chceme-li vytvořit fotografii z fotoaparátu a poté ji nějak zpracovat, vytvoříme Intent pro vytvoření fotografie, OS na to zareaguje spuštěním aplikace fotoaparátu, a po vyfocení se vrátíme do naší aplikace a dostaneme objekt s vyfocenou fotografií. [2][3]

Intent ale také slouží pro přepnutí na jinou Activity v naší aplikaci. V tomto případě se založí přímo s názvem třídy Activity kterou chceme spustit.

Tento záměr nebo úmysl reprezentuje třída `android.content.Intent` a pomocí funkce `Intent setData(Uri data)` jí lze předat data pro splnění úmyslu. Na obrázku Obrázek 5 je část zdrojového kódu, který provádí záměr vytočit dané telefonní číslo. Ten pokud je nainstalována telefonní aplikace, spustí vytáčení čísla.

```
1. Intent callIntent = new Intent(Intent.ACTION_CALL);
2. callIntent.setData(Uri.parse(("tel:123456789"))
3. startActivity(callIntent)
```

Obrázek 5 Záměr vytočit telefonní číslo

Které Intent bude Vaše Aktivita přijímat se definuje pomocí intent filterů v souboru AndroidManifest.xml. Pokud záměr projde vašim filterem (Podle typu, dat, apod.) bude vaše aktivita v nabídce pro splnění tohoto záměru.

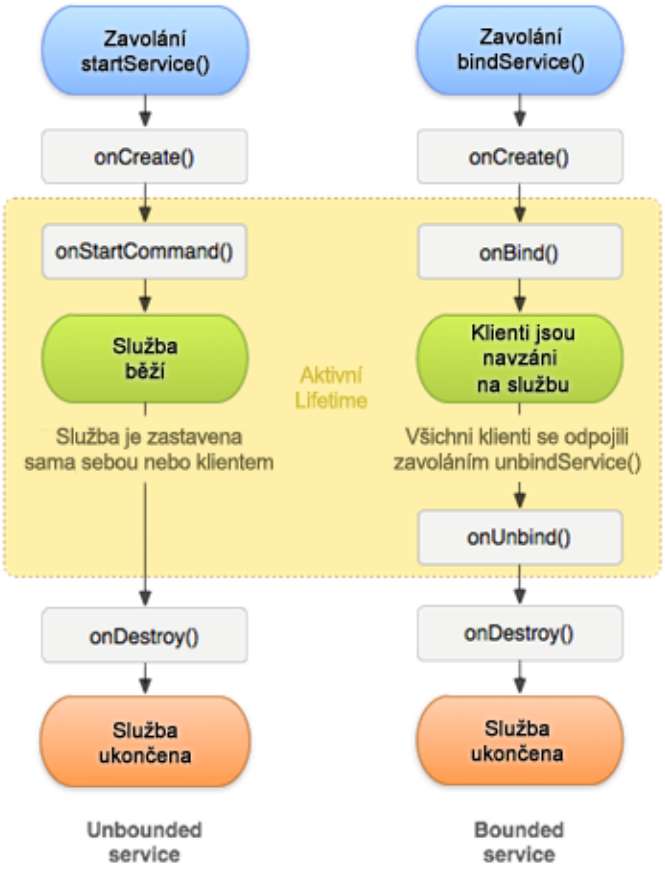
2.4.5 Service

Aplikační Framework Androida poskytuje samozřejmě kromě klasických obrazovek aplikace, také služby, které běží na pozadí a neposkytují žádné uživatelské rozhraní. Ty se označují Service. Tyto služby se používají především k vykonávání dlouho trvajících operací nebo pro přístup ke vzdáleným zdrojům, kde může být dlouhá doba odezvy. Vlastní Service dědí od třídy `android.app.Service` a dá spustit dvěma způsoby. [10]

1. **Nastartováním** pomocí metody `startService()`. V tomto případě po té co vykoná svou činnost se sama ukončí nebo ji může ukončit jiná komponenta.
2. **Navázáním** pomocí metody `bindService()`, kterou zavolá jinou komponenta, čímž se stává klientem této služby. V tomto případě se služba ukončí po odpojení všech klientů a nebo ji může ukončit klient, který ji založil.

2.4.6 Broadcast reciever

Díky Broadcast recieverům je možné reagovat na systémové události, jako jsou například rozsvícení displeje, přijatý hovor, přijatá SMS, ztráta připojení k internetu, nízký stav baterie a mnoho dalších. Stejně tak naše vlastní Activity nebo Service může vyvolat vlastní Broadcastovou zprávu, kterou náš (nebo jiný) Broadcast reciever může přijmout. Stejně jako u Service tato komponenta systému nemá žádné uživatelské rozhraní. Po přijetí zprávy může vytvořit například notifikaci, nebo v krajním případě otevřít některou Activity. Vlastné Broadcast recivery dědí od třídy `android.content.BroadcastRevicer` a implementují metodu `onReceive(Context context, Intent intent)`.[1][3]



Obrázek 6 Životní cyklus Service; Zdroj [10]; provedení vlastní

2.4.7 Content Provider

Poslední neméně důležitou komponentou pro tvorbu aplikací je Content Provider. Jak název napovídá, slouží pro poskytování obsahu. A to jak pro poskytování dat ostatním aplikacím, tak pro poskytování dat pouze mezi vlastními Activity. Content Provider dědí od třídy `android.content.ContentProvider` a poskytuje jednoduché rozhraní s metodami

`query()`, `insert()`, `update()` a `delete()`. Muže je implementovat například jako ukládání dat do lokální SQLite databáze nebo na server pomocí REST API.

Mezi systémové Content Providery patří například čtení / zápis / dotazování se / mazání z Kalendáře, Uživatelského slovníku, Kontaktů apod. Odkud chceme data číst určuje URI začínající protokolem content. Po něm následuje unikátní identifikátor aplikace, který se skládá z jejího *namespace*. Např. `content://com.example.notepad.provider/notes` je adresa provideru aplikace z balíčku `com.example.notepad` a pracovat se bude s „tabulkou" notes. Čtení z Content Provideru poté obstarává Content Resolver, který se k této URI připojí a pracuje s metodami provideru. [2][3]

3 Google Cloud Messaging

Oficiální cesta jak vzdáleně posílat z libovolné ho vašeho server vaší aplikace zprávy na mobilní zařízení přes internet je využití služby Google Cloud Messaging. GCM je bezplatná služba pomáhající vývojářům odesílat data ze serveru do jejich aplikace běžící na zařízení s Android OS a naopak zasílat zprávy ze zařízení zpátky na cloud a z něj poté na server.

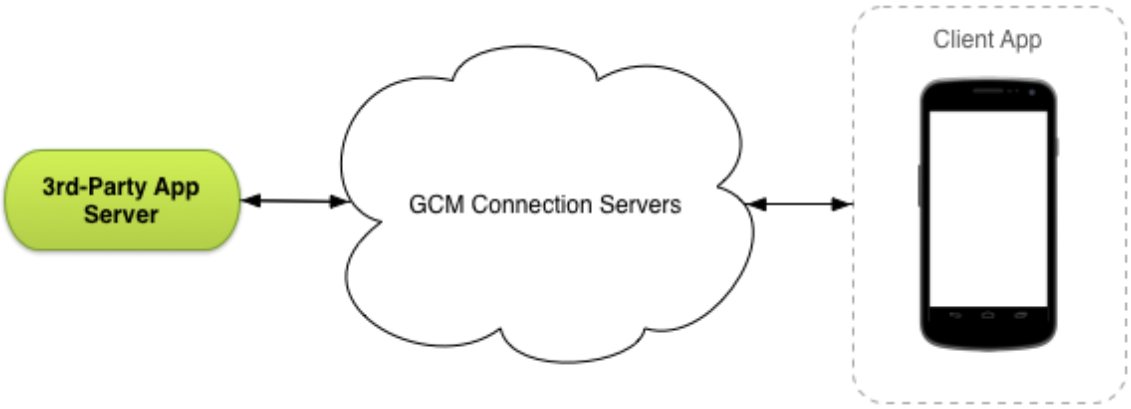
Tyto zprávy můžou být pouze oznámení o tom, že si má aplikace ze serveru něco stáhnout (například notifikace, že na server přišel nový email) nebo zprávy s daty až do 4kB. Takže aplikace zajištující třeba Instant Messaging může GCM používat přímo pro přenos finálních dat (zpráv). GCM také řeší frontu zpráv v případě, že je zařízení vypnuté a seskupování zpráv stejného typu aby přišla vždy jen ta nejaktuálnější. [11]

3.1 Architektura a životní cyklus zasílání zpráv

Implementace GCM zahrnuje server provozovaný Googlem, který zprostředkovává zasílání zpráv mezi serverem 3. strany a aplikací v Android mobilním zařízení. Tento server přijme zprávu od aplikačního serveru (serveru 3. strany) buďto pomocí XMPP protokolu nebo pomocí HTTP protokolu. GCM Connection server má také za úkol uchovávat zprávy ve frontě, pokud je nemůže hned předat klientské aplikaci. Po připojení klientského zařízení k internetu zašle GCM server zprávy na zařízení.

Klientská aplikace je Android aplikace s povoleným GCM běžící na některém zařízení. Pro příjem GCM zpráv musí být aplikace registrovaná u GCM a musí získat registrační ID. Pokud se pro komunikaci mezi serverem a GCM používá protokol XMPP, může také aplikace posílat zprávy přes GCM přímo na server. [11]

Pro používání služby Google Cloud Messaging je nejprve nutné ji u Googlu aktivovat. To se dělá v Google Developers konzoli na webové adrese https://console.developers.google.com. Zde se založí nový projekt, který dostane přiřazeno své číslo, tzv. Project Number. Toto číslo poté slouží k adresování, když zařízení posílá zprávu na server. Dále je zde také potřeba vytvořit API KEY, který se používá pro autorizaci požadavků.



Obrázek 7 Schéma komunikace přes GCM, Zdroj [11]

3.1.1 Registrace zařízení

Při prvním spuštění aplikace na zařízení je potřeba aplikaci nejprve zaregistrovat na GCM serveru. To se provádí pomocí metody `register()` na objektu třídy `GoogleCloudMessaging`. Metoda `register()` vrátí tzv. registrační ID které je potřeba uložit v zařízení pro další použití a také je potřeba zaslat ho na server 3.strany, protože tímto ID se adresuje zařízení na které bude server zasílat zprávy.

3.1.2 Zaslání zprávy

Posloupnost akcí při zaslání zprávy z aplikačního serveru na aplikaci v zařízení je následující:

1. Aplikační server zašle zprávu na GCM server.
2. Google zařadí zprávu do fronty a uloží ji v případě, že je zařízení off-line.
3. Jakmile je zařízení online, Google pošle zprávu na zařízení
4. Na zařízení systém zajistí, aby zpráva došla konkrétní aplikaci a žádné jiné. Tato aplikace musí mít povolená příslušná oprávnění. Systém při přijetí zprávy „probudí aplikaci“, takže do té doby nemusí být vůbec spuštěná.
5. Aplikace zpracuje zprávu. Pokud to má znamenat nějaký náročnější výpočet, je vhodnější provést výpočet pomocí služby na pozadí.

Aplikace může odregistrovat GCM, pokud už nebude potřeba dále přijímat zprávy. [11]

3.1.3 Přijetí zprávy

Posloupnost akcí při přijetí zprávy na zařízení:

1. Systém přijme zprávu a přečte z ní dvojice klíč-hodnota, které jsou v datech zprávy.
2. Systém se předá dvojice klíč-hodnota cílové aplikaci pomocí extra dat Intentu `com.google.android.c2dm.intent.RECEIVE`
3. Aplikace si převezme data z Intentu a zpracuje je. [11]

3.2 Implementace na klientovi

V android aplikaci se vše spojené s komunikací provádí pomocí třídy `GoogleCloudMessaging` z balíčku `com.google.android.gms.gcm`. Aby tato knihovna byla dostupná je potřeba ji nejdříve připojit pomocí balíčkovacího systému Gradle, a to dopsáním závislosti do souboru build.gradle. [11]

```
dependencies {
  compile "com.google.android.gms:play-services:3.1.+"
}
```

Obrázek 8 Přidání závislosti do projektu

Dále je potřeba do souboru android manifest dopsat několik oprávnění souvisejících se službou GCM:

- Oprávnění `com.google.android.c2dm.permission.RECEIVE` povolující aplikaci registrovat se u GCM a přijímat GCM zprávy.
- Oprávnení `android.permission.INTERNET` povolující využít internet pro zaslání svého Registration ID na server 3. strany.
- Oprávnění `android.permission.WAKE_LOCK` povolující probuzení procesoru z režimu spánku když přijde zpráva.
- Speciální oprávnění <název balíčku aplikace>.permission.C2D_MESSAGE, které zabrání ostatním aplikacím v přijímání zpráv určených této aplikaci.

3.3 Implementace na serveru

Nejprve je potřeba vybrat si mezi dvěma servery, které Google nabízí pro GCM. HTTP server nebo CCS (XMPP) server. Hlavní rozdíl je především v možnosti posílat zprávy i ze zařízení na server, ale také v synchronním/asynchronním zasílání zpráv nebo možnosti zaslání zprávy na víc zařízení na jednou.

Ať už použijeme kterýkoli z těchto protokolů, budeme na server zasílat nějaké zprávy a budeme očekávat odpověď, jestli vše proběhlo v pořádku. Popis nejdůležitějších z nich je vidět zde v tabulce Tabulka **3** a zbytek poté v dokumentaci [11].

Parametr	Popis	Podporováno
`to` *(povinný)*	V CCS se tento parametr používá místo `registration_ids` pro určení příjemce zprávy. Jedná se právě o to ID, které je přiděleno v klientské aplikaci při registraci zařízení do GCM.	CCS
`message_id` *(povinný)*	Tento parametr jednoznačně identifikuje zprávy v souvislosti s XMPP.	CCS
`message_type` *(volitelný)*	Zde se uvádí typ zprávy. Jestli jde o data, ACK nebo NACK zprávu.	CCS
`registration_ids` *(povinný)*	Tento parametr určuje pole obsahující seznam registračních ID příjemců. Musí obsahovat alespoň 1 a nejvýše 1000 ID.	HTTP
`collapse_key` *(volitelný)*	Tento parametr obsahuje libovolný řetězec, který se používá pro seskupování zpráv stejného významu, když je zařízení off-line. Po připojení klienta se tak pošle vždy jen nejnovější aktualizace.	CCS, HTTP

`data`
(volitelný)
	Do tohoto pole posíláte vlastní data ve dvojicích Klíč-Hodnota které potom zpracuje aplikace na zařízení. V JSONu například:
	`{`
	`type: "new_email",`
	`from: somebody@mail.tld`
	`date: "2014-11-06 09:13:00"`
	`}`
		CCS, HTTP

`time_to_live` *(volitelný)*	Tento parametr určuje jak dlouho (v sekundách) by měla být zpráva uchována na server GCM, v případě že je zařízení off-line. Výchozí TTL je 4 týdny.	CCS, HTTP
`restricted_package_name` *(volitelný)*	Tento parametr určuje název balíčku aplikace. Pokud je nastaven, zprávy jsou doručený jen	HTTP

	na registrační ID, kde název balíčku aplikace odpovídá tomuto.	
dry_run	Tento parametr slouží pro vývojáře pro testování. Zpráva se ve skutečnosti neodešle, ale odpověď bude stejná jako kdyby ano.	HTTP

Tabulka 3 Parametry zprávy od serveru 3.strany k GCM serveru; Zdroj [11]

3.3.1 HTTP server

Jeho implementace je značně jednoduší. Stačí, například pomocí CURL, zaslat HTTP POST požadavek na URL adresu https://android.googleapis.com/gcm/send/. Vzhledem k tomu, že HTTP požadavek je blokující a musí se čekat na odpověď dřív než se zašle nová zpráva, označuje se tento přístup za synchronní. Další nevýhodou je, že lze zprávy zasílat jen od serveru k zařízení a ne naopak. To se případně musí řešit jiným způsobem. Autentizaci se v tomto případě využívá HTTP hlavička `Authorization` s hodnotou `key=<přidělený API klíč>`.

Při komunikaci pomocí HTTP je také možné zvolit dva způsoby formátu dat. Buď se dá použít JSON, který je potřeba specifikovat hlavičkou `Content-Type: application/json` v požadavku a nebo lze použít Plain text. V tomto případě je možné hlavičku Content-type vypustit, případně použít s hodnotou `application/x-www-form-urlencoded;charset=UTF-8`. Při použití plain textu bohužel nejde zadat více příjemců zprávy. [11]

```
Content-Type:application/json
Authorization:key=AIzaSyB-1uEai2WiUapxCs2Q0GZYzPu7Udno5aA

{
  "collapse_key": "score_update",
  "time_to_live": 108,
  "delay_while_idle": true,
  "data": {
    "score": "4x8",
    "time": "15:16.2342"
  },
  "registration_ids":["4", "8", "15", "16", "23", "42"]
}
```

```
Content-Type:application/x-www-form-urlencoded;charset=UTF-8
Authorization:key=AIzaSyB-1uEai2WiUapxCs2Q0GZYzPu7Udno5aA

collapse_key=score_update&time_to_live=108&delay_while_idle=1&data.scor
e=4x8&data.time=15:16.2342&registration_id=42
```

Obrázek 9 JSON a PlainText HTTP požadavek

3.3.2 CCS (XMPP) server

GCM Cloud Connection Server je XMPP server, který poskytuje persistentní, asynchronní a obousměrné připojení ke Google serveru. Komunikaci pomocí XMPP protokolu používají různé Instant Messaging služby jako například Jabber nebo Facebook chat. Komunikace s Google serverem probíhá obdobně, jako když si s někým píšete zprávy. Tím zajišťuje zasílání zpráv z aplikačního serveru do zařízení, tak i zaslání zprávy ze zařízení k aplikačnímu serveru. CCS je upravená varianta XMPP, která zapouzdřuje data v JSON formátu a má nějaké drobné změny ve formátu XML požadavků.

Nejprve se ustálí připojení mezi CCS a vaším aplikačním serverem (který v tomto případě zastává roli XMPP klienta). Dále se zasílají XMPP správa s data zakódovanými v JSONu. Vzhledem k tomu, že je potřeba držet persistentní spojení a udržovat asynchronní datový tok, musí v tomto případě běžet server 3. strany pořád. Pro implementaci CSS klienta je vhodný například Python nebo Java, kde aplikace běží po celou, dobu dokud ji nevypneme, což by se třeba v PHP řešilo dost nevhodně. [11]

4 Analýza konkurenčních řešení

Aplikací pro komunikaci se ztraceným mobilním zařízením přes internet je mnoho. V této kapitole představím ty nejznámější z nich a uvedu jejich hlavní funkce. Existuje originální aplikace přímo od Googlu ale i mnoho dalších např. od známých firem zabývajících se bezpečností a antivirovými systémy. Google Play ovšem obsahuje i spoustu aplikací, které na první pohled vypadají nedůvěryhodně.

Každá aplikace, které uživatel schválí administrátorská a jiná nebezpečná práva může být zdrojem potencionálních bezpečnostních problémů, jako jsou například tzv. „zadní vrátka“. Proto je potřeba důvěřovat společnosti, která aplikaci vyvíjí.

Nyní si představíme mnou vybrané aplikace pro komunikaci se ztraceným telefonem

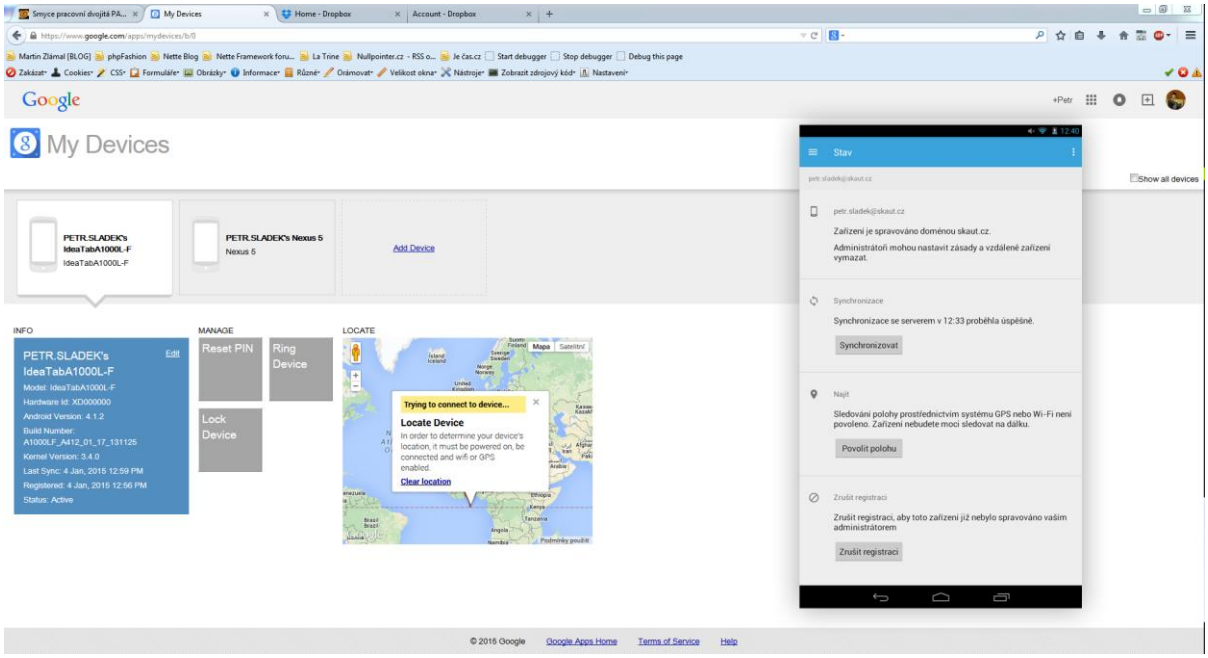
4.1 Google Apps Device Policy

Jedná se o originální aplikaci, kterou provozuje Google. Webová aplikace je dostupná na adrese https://www.google.com/android/devicemanager. Propojení mobilní a webové aplikace probíhá pomocí Google účtu automaticky. Po nainstalování aplikace je potřeba potvrdit jako správce zařízení a povolit jí administrátorská práva (pro nastavování hesel apod.)

Webová aplikace je jednoduchá ve stylu Googlu. Dají se zde přepínat jednotlivá zařízení a na nich provádět následující akce:

- Vypsat informace o zařízení (model, sériové číslo, verze kernel, datum poslední synchronizace, atp..).

- Nastavit/Změnit PIN kód pro přístup k zařízení.
- Rozezvonit zařízení.
- Zamknout telefon. Pokud není nastaven PIN nebo Gesto pro odemknutí tak téměř nemá význam.
- Zjistit polohu zařízení a zobrazit ji na mapce.



Obrázek 10 Google Device Policy webová a mobilní aplikace

Celkově je aplikace velmi jednoduchá. Pro základní nalezení a zamknutí telefonu ovšem postačí. Bohužel nenabízí žádnou možnost případnému nálezci telefonu dát vědět, že byl telefon nalezen (třeba po delší době, kdy už na něj majitel přestal sám volat).

4.2 AndroidLost

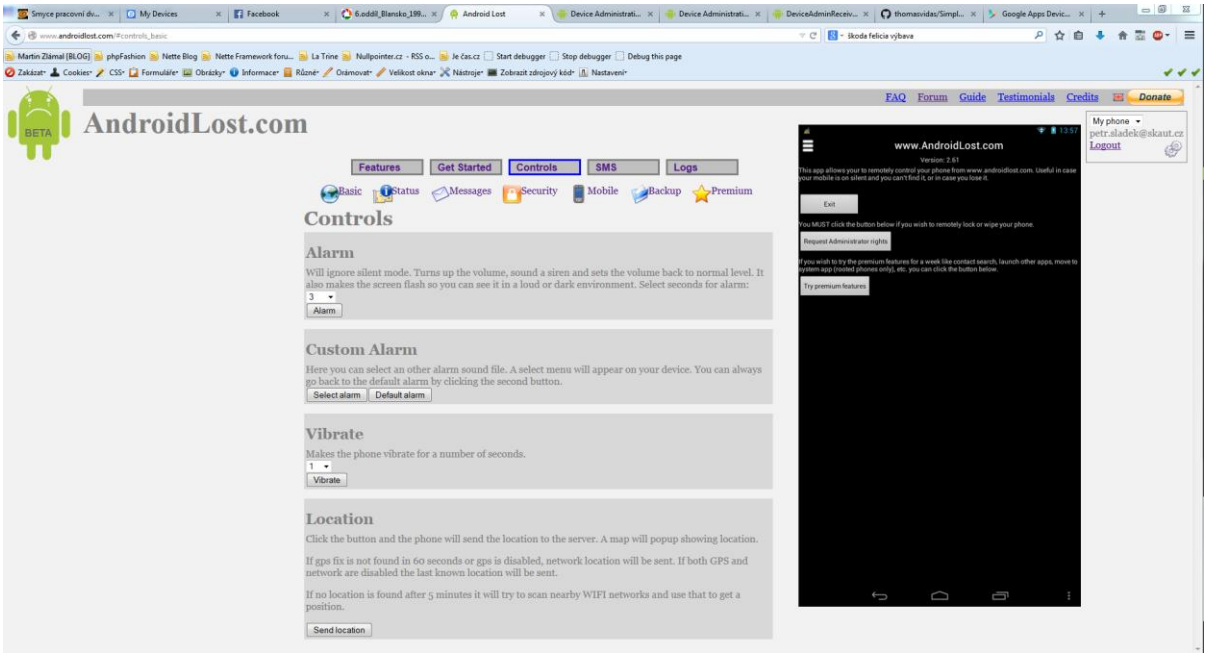
Toto je provedením velmi jednoduchá aplikace od Dánského autora jménem Theis Borg. Přestože je stále označována jako Beta verze, má mnoho funkcí a dokonce má i placenou prémiovou variantu, která ještě další funkce přidává. Webová aplikace je dostupná na adrese http://www.androidlost.com/ a je přihlásit se opět stačí pouze pomocí Google účtu, který je i v mobilním zařízení.

Mobilní aplikace jde oproti ostatním ovládat i pomocí SMS zpráv. Což je velká výhoda v případě, že ztracené zařízení nemá přístup na internet (například vůbec nemá datový tarif). Další zajímavá vlastnost, oproti předchozí Google Device Policy, je že v Launcheru v má ikonku jako poznámkový

blok a název notepad. To může zabránit proti tomu, aby nezkušený zloděj, který by se například dostal k odemčenému telefonu, tuto aplikaci odinstaloval.

Funkce, které aplikace podporuje, jsou následující:

- Vyvolat alarm na několik sekund (přičemž ignoruje tichý mód).
- Vyvolat vibrace na několik sekund.
- Zjistit a ukázat na mapě lokaci zařízení.
- Zjistit stav zařízení (IMEI, stav baterie informace o operátorovi a SIM kartě, informace o WiFi a mnohé další).
- Zapnout / vypnout tichý mód.
- Zapnout / vypnout Bluetooth (například pro odpojení od headsetu, aby se dal mobil po zvuku najít).
- Rozsvítit / zhasnout LED diodu (tzn. blesk od fotoaparátu).
- Zapnout / vypnout GPS a WiFi (nemusí ovšem fungovat na všech zařízeních).
- Stáhnout seznam aplikací, posledních SMS či posledních volání.



Obrázek 11 AndroidLost webová a mobilní aplikace

- Poslat na zařízení textovou zprávu, která se zobrazí jako popup okno.
- Poslat textovou zprávu a po kliknutí uživatele na OK zaslat zpět fotku z přední kamery.
- Nastavit textovou zprávu, která se zobrazí po spuštění telefonu (například s větou zavolejte mi na číslo +420 123 456 897).
- Nastavit textovou zprávu, která se bude červně zobrazovat na LockScreenu.

- Oznámit, že SIMkarta v telefonu není moje. Poté lze opět zasílat opět zasílat SMS příkazy, i když někdo oddělá můj Google účet.
- Nastavit / zrušit PIN kód.
- Vymazat SD kartu.
- Vymazat telefon a vrátit do továrního nastavení.
- Zaslat textovou zprávu, která se přečte (tzv „text to speech“).
- Vytočit telefonní číslo (neboli zavolat si).
- Zaslat fotku z přední / zadní kamery.
- Nahrát několik sekund zvuku.
- Zaslat screenshot.
- Vypnout / Restartovat telefon (vyžaduje mít tzv. rooted telefon).

Jak je vidět tato aplikace poskytuje mnoho jednoduchých příkazů, které může uživatel použít, ale žádné komplexnější řešení pro dané situace. Proto bych ji doporučil spíše pokročilým uživatelům.

4.3 Where’s My Droid

Tato aplikace, na rozdíl od minulé, má složitější nastavení na straně telefonu a vcelku jednoduše udělanou webovou aplikaci. Například na webu se dá volba rozezvonit telefon, ale jak dlouho bude zvonit, jestli u toho bude vibrovat a jestli se rozsvítí LED dioda blesku, se už musí nastavit dopředu v aplikaci.

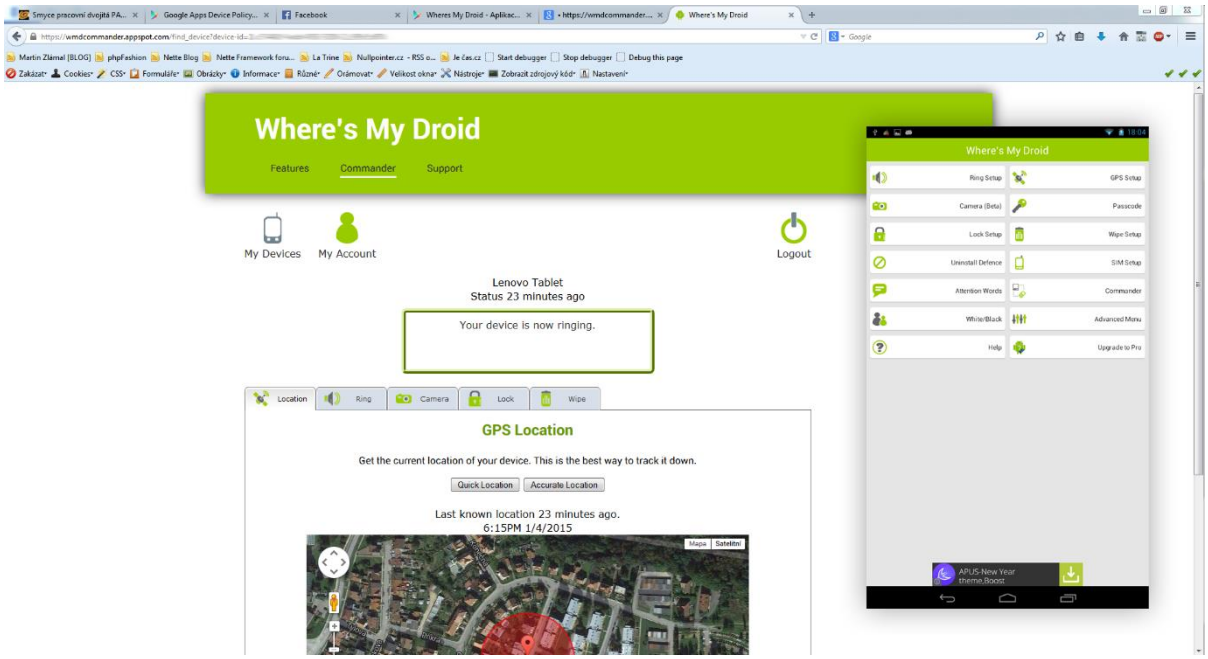
Nevýhodou je, že si uživatel musí ve webové aplikaci dostupné na https://wmdcommander.appspot.com/ nebo v mobilní aplikaci vytvořit nový účet. To mi v dnešní době připadá už velmi přežité. Bývá dobrým zvykem možnost se přihlašovat pomocí Google účtu, Facebooku, Twitteru či přes OpenID.

Stejně jako u minulé aplikace, lze jednotlivé funkce vyvolávat pomocí příkazů v SMS. Texty SMS si jde dokonce sám nastavit a jednotlivé funkce pro SMS nebo Internet aktivovat/deaktivovat. Také lze nastavit čísla, která mohou nebo nemohou SMS příkazy zasílat. Funkce, kterými aplikace disponuje, jsou následující:

- Spustit alarm (melodie, vibrace, blikání).
- Vyfotit fotku z přední / zadní kamery (pouze v placené verzi).
- Zjistit polohu zařízení.
- Nastavení / Zrušení PIN kódu.
- Vymazání SD karty.
- Uvedení telefonu do továrního nastavení.

- Nastavení ochrany proti odinstalování, zadáním speciálního PIN kódu (pouze v placené verzi).
- Lze skrýt / zobrazit ikonku aplikace v Launcheru.
- Lze nastavit zaslání upozornění v případě změny SIM karty.

Celkově mi přijde, že aplikace nemá ve verzi zdarma moc užitečných funkcí a také je plná reklam.



Obrázek 12 Where's My Droid webová a mobilní aplikace

4.4 Theftie – Find my phone

Mobilní i webová aplikace dostupná na http://www.theftie.net/ je už na první pohled hezky zpracována a působí velmi věrohodně. Pro uživatele, jak sami autoři uvádí [23], má tři hlavní funkce:

1. Zabránit: Zjistit neoprávněný přístup do telefonu a okamžitě uzamknout přístroj a poslat upozornění na Váš E-mail.
2. Ulovit: Zjistit umístění zařízení na mapě, a fotku uživatele z přední kamery. Dále je možné zasílat na telefon zprávy, u kterých je také možné zobrazit tlačítko „Zavolejte mi zpět“.
3. Záchrana dat: Zálohuje dokumenty a fotografie na Váš Google disk.

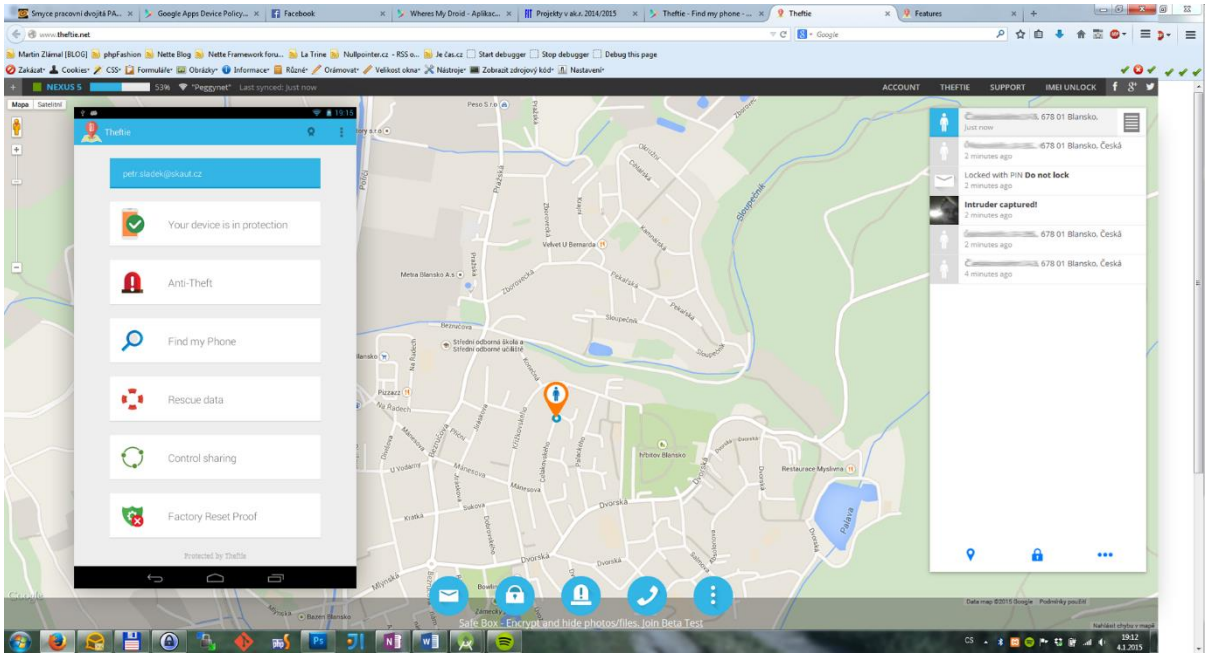
Tato aplikace je asi nejvíce cílena na běžného uživatele. Opět lze zasílat příkazy i pomocí SMS. Aplikace má tzv. LockMode, kdy zobrazí na telefonu zamykací obrazovku, s tlačítky odemknout (pro zadání specifického PINu) a zavolat mi zpátky. Dále má skvělou vlastnost, že do konfigurace aplikace se dá dostat také až po zadání PINu, který si definujete při prvním spuštění. Pomocí příkazů z webové

aplikace je také možné skrýt či zobrazit ikonu ke spuštění aplikace na telefonu. Náhled obrazovky ve webovém prohlížeči a v mobilní aplikaci je vidět na Obrázek 13.

5 Specifikace a návrh aplikace

V této kapitole je uvedeno, co bude aplikace umět a jak bude vypadat. V první části se pokusím navrhnout způsoby jak má aplikace pomoci majiteli mobilního zařízení k jeho opětovnému nalezení. Navrhnu způsoby jak v různých situacích mobil lokalizovat či zablokovat, případně jak usnadnit případnému nálezci navrácení telefonu do správných rukou.

V druhé části se budu zabývat návrhem mobilní aplikace pro Android a webové aplikace, která poběží na veřejném webovém serveru.

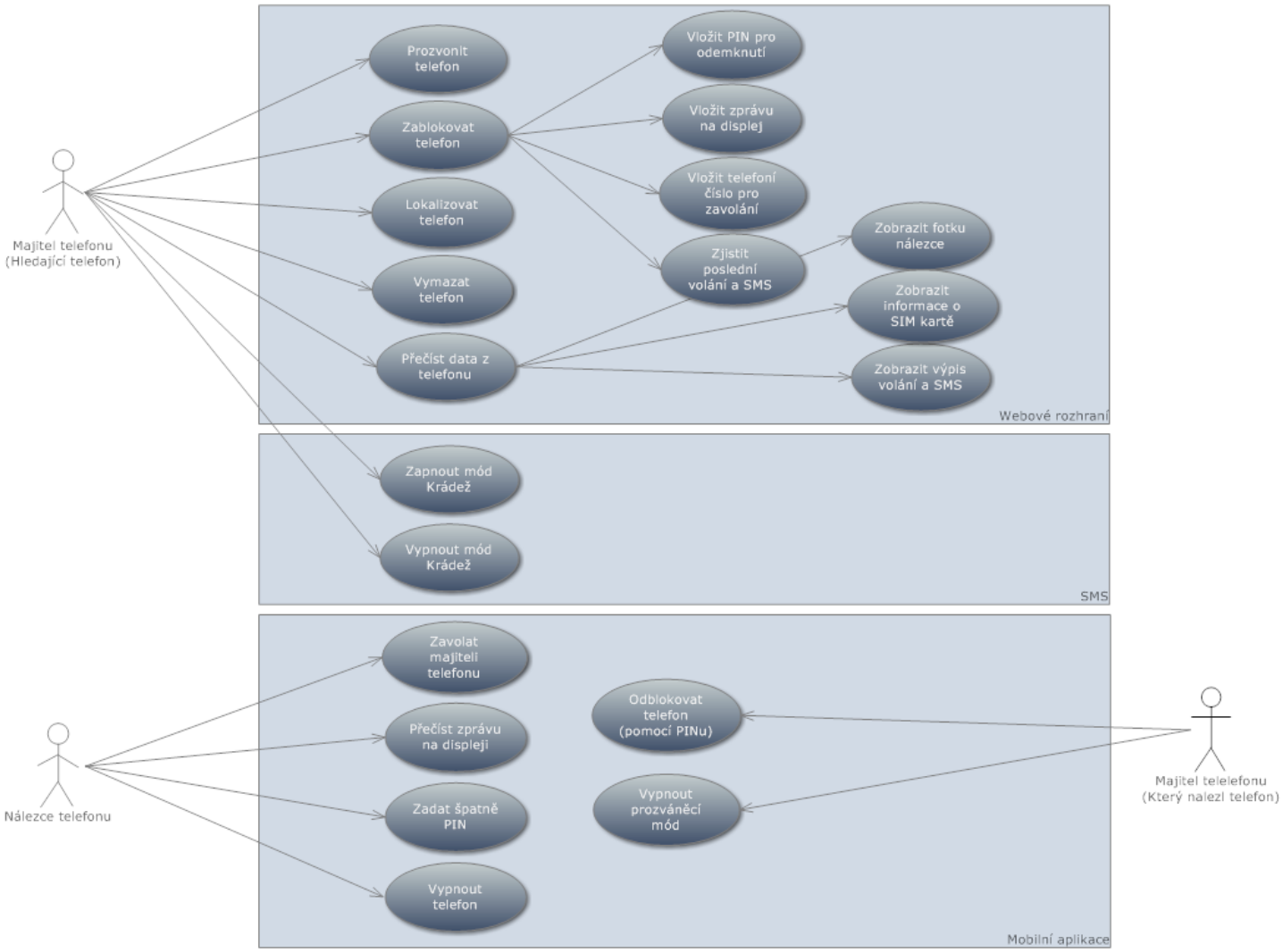


Obrázek 13 Theftie webová a mobilní aplikace

5.1 Specifikace požadavků

Moje aplikace nebude vykonávat v mobilním zařízení jen dílčí akce jako je zapnout/vypnout vibrace apod., ale bude řešit komplexní scénáře, které mohou v reálné situaci nastat. A to především z důvodu aby se s ní i uživateli laikovi lépe pracovalo. Tímto přístupem se asi bude nejvíce podobat aplikace Theftie z předchozí kapitoly. Bude ovšem oproti ní nabízet některé funkce jinak nebo navíc.

Nyní se podíváme na jednotlivé scénáře, které mohou nastat v situaci se ztraceným telefonem. U každého scénáře uvedu, způsob jak aplikace pomůže, abychom telefon mohli opět nalézt. Samozřejmě někdy se můžeme dostat do situace, kdy nevíme, kde jsme telefon zapomněli, nebo jestli nám ho někdo ukradl. V tom případě nezbývá nic jiného než vyzkoušet všechny možnosti.



Obrázek 14 Use case diagram aplikace

5.1.1 Scénář: ztracený telefon např. v bytě nebo autě

Toto zná zřejmě každý. Telefon je pravděpodobně někde v kapse, položený na nějakém neobvyklém místě, zapadlý za postelí nebo Vám vypadl v autě z kapsy pod sedadlo. K tomu, zjistit kde přesně je, poslouží naše aplikace.

V tomto případě by po vstupu na webovou část aplikace mělo být hned k dispozici tlačítko „*Prozvonit telefon*“. To i v případě ztlumeného zvuku na telefonu rozezvoní, rozvibruje a rozbliká telefon. A to do doby, než se dalším tlačítkem na webu, nebo na telefonu tato akce ukončí.

Dále by mělo tato akce deaktivovat Bluetooth, aby v případě připojeného headsetu, telefon zvonil opravdu sám a dal se v bytě, autě, či garáži bez problémů najít. V případě, že akci dlouho nikdo nedeaktivuje, deaktivuje se po vypršení daného času sama, a mezi tím se telefon pomocí lokalizační služby pokusí nalézt přibližnou polohu zařízení (pro potvrzení jestli je telefon opravdu doma).

Může nastat situace, že mobilní telefon bude mít vybitý akumulátor nebo bude jen vypnutý. V tom případě bohužel neexistuje řešení.

5.1.2 Scénář: ztracený telefon, který nikdo nenašel

V případě, že víme, že jsme mobilní telefon někde ztratili nebo zapomněli, nastává úplně jiná situace. Bude potřeba mobil uzamknout, aby se do něj v případě nálezu nikdo nedostal. A především v případě, že na telefonu nemáme aktivováno odemykání PINem ani odemykání Gestem.

Dále bude potřeba telefon lokalizovat, abychom alespoň přibližně věděli, kde se nachází. A mohli pro něj případně přijet. Zabýváme se zde situací, že je telefon ztracený například někde v lese, či mimo civilizaci, takže ho pravděpodobně nikdo nenajde. Jakmile se dostaneme do místa, kde telefon přibližně je, použijeme scénář z předchozí kapitoly. Bude ovšem potřeba dát pozor, aby náhodou nenalezl telefon dříve někdo jiný, kdo by si ho mohl chtít nechat.

Pro tento scénář bude webové rozhraní obsahovat tlačítka pro akce „*Uzamknout telefon*“, „*Lokalizovat telefon*“ a tlačítko z minulého scénáře „*Prozvonit telefon*“. V momentě uzamknutí telefonu se zadá PIN, který poté bude potřeba pro následné odemknutí telefonu. Funkce lokalizovat telefon nám ukáže jeho poslední polohu na mapě.

Uzamykací obrazovka by v případě vypnutého telefonu měla naskočit ihned po jeho zapnutí. To pro případ, kdyby telefon někdo našel a zapnul ho (v případě vybytí připojil na nabíječku a zapnul). O tomto scénáři více v další kapitole.

5.1.3 Scénář: ztracený telefon našel ten, kdo ho chce vrátit

Tento scénář je podobný předchozímu. V tomto případě ovšem předpokládáme, že telefon někdo nalezne a bude nám ho chtít navrátit. Proto aplikace musí umět usnadnit co nejvíce případnému nálezci komunikaci s majitelem zařízení.

Proto po uzamknutí telefonu, bude na uzamykací obrazovce, kde se zadává PIN kód pro odemknutí ještě tlačítko „Zavolat majiteli“. Telefonní číslo, které se bude vytáčet se zadá ve webovém rozhraní hned po akci *uzamknout telefon*.

Dále bude možné přidat zprávu, která se na mobilním zařízení zobrazí u tohoto tlačítka. Majitel telefonu sem může z webu zadat vlastní text. Např.: „Za vrácení telefonu nabízím odměnu 2000 Kč“ nebo prostě „V případě nezlezení tohoto telefonu mi prosím zavolejte zpátky“. Tyto zprávy by mělo webové rozhraní nabízet, a uživatel je bude moci upravit, nebo si napsat úplně vlastní.

5.1.4 Scénář: ztracený telefon našel ten, kdo si ho chce nechat, nebo ho někdo ukradl

V tomto případě opět navazujeme na předchozí dva scénáře. Ovšem zde předpokládáme, že případný nálezce telefon vrátit nechce, a chce si ho buď nechat, nebo ho prodat. Pro majitele telefonu je to bohužel nejhorší scénář, protože hrozí, že už telefon nikdy neuvidí.

Stejně jako v minulých případech půjde telefon lokalizovat a uzamknout. Při tomto scénáři ovšem ještě přibydou další funkce. V případě, že nálezce nezareaguje na výzvu zavolat majiteli zpět, a pokusí se tipnout PIN kód, telefon ho vyfotí přední kamerou a pošle fotku na webový server. A to při každém špatném zadání PINu. Stejně tak vyfotí přední kamerou osobu při vypnutí telefonu tlačítkem. V případě, že je telefon mimo signál, fotografie se pošlou ihned při připojení k síti.

Když někdo v telefonu vymění SIM kartu, opět se pošle na webový server zpráva, u které budou připojeny všechny zjistitelné údaje o SIM kartě. Tyto nové funkce by měli pomoci nalézt pachatele krádeže / nálezce telefonu. V případě krádeže bude ovšem potřeba kontaktovat Policii.

Ještě se nabízí možnost, že dlouho nezjistíte, že vám telefon chybí a proto ho nestihnete zamknout. V případě, že by v této chvíli pachatel telefon použil, půjdou přes webové rozhraní zjistit poslední volaná čísla a přijaté/odeslané SMS zprávy.

5.1.5 Scénář: ukradený telefon z kapsy

Poslední scénář jak přijít o telefon je, že vám ho někdo ukradne z kapsy, vy to ihned zjistíte, ale pachatele se vám nepodaří ve velkém davu identifikovat, či Vám uteče. V tomto případě nemáte u sebe počítač s internetem, ale budete mít poblíž například kamaráda s mobilním telefonem. Aplikace proto bude umět zareagovat na příkaz pomocí SMS zprávy.

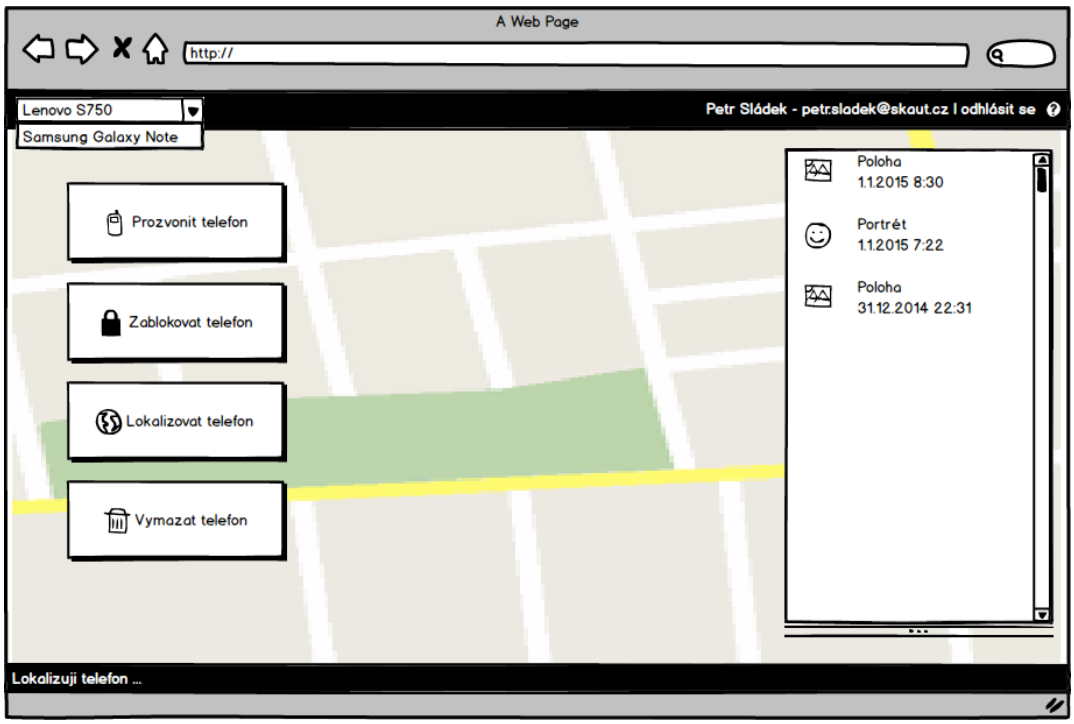
Tento SMS příkaz uzamkne telefon, stejně jako to bylo v předchozích scénářích, plus ještě k tomu začne na hlas přehrávat zprávu „Tento telefon byl právě ukraden“ a do toho pískat sirénou. To by mělo přinutit pachatele telefon minimálně zahodit a poté můžeme pokračovat podle předchozích scénářů.

5.1.6 Scénář: Telefon už pravděpodobně nenalezneme

Zde už jsme se smířili s tím, že telefon zpět nedostaneme a proto ho budeme chtít alespoň vymazat. Z webového rozhraní proto půjde *vymazat SD kartu* a nebo případně celý *telefon uvést do továrního nastavení*. Toto je ovšem poslední akce, kterou budeme moci provést, protože po tomto kroku se smaže i naše aplikace.

Pokud se Váš telefon dostane do ruky profesionálovi, který ihned telefon připojí k počítači a celý ho nějak vymaže a uvede do továrního nastavení sám, tak naše aplikace bohužel už nijak napomůže.

Akorát už od registrace budeme vědět IMEI kód zařízení (který je mimo jiné i v dokladech při koupi telefonu). Ten budeme moci nahlásit polici při ohlášení krádeže.



Obrázek 15 Wireframe obrazovky webové aplikace po přihlášení

5.2 Návrh webové aplikace

Cílem této kapitoly uvést návrh vlastního řešení webové aplikace pro komunikaci se ztraceným mobilním telefonem, podle kterého poté bude implementována. V následujících podkapitolách bude uveden návrh grafického rozhraní, databázové struktury webové aplikace a v neposlední řadě také návrh způsobu komunikace mezi webovou a mobilní aplikací.

Pro implementaci webové aplikace jsem se rozhodl použít na straně serveru PHP ve verzi 5.4, které má oproti starším verzím mnohé novinky v objektově orientovaném modelu. Dále pak MySQL databázi pro ukládání dat. Z mnohých PHP frameworků jsem vybral Nette, které má dle mých zkušeností nejlépe vyřešené API jednotlivých tříd a používá tzv. Dependenci injection. Na straně prohlížeče bude webová aplikace napsaná v HTML5 s využitím JavaScriptu.

5.2.1 Návrh grafického rozhraní

Webová aplikace bude místem jen pro majitele telefonu. Proto je nutné, aby se do ní přistupovalo přes login. Přihlašovat se půjde jen pomocí Google účtu, což zajistí přímo spárování s mobilními telefony

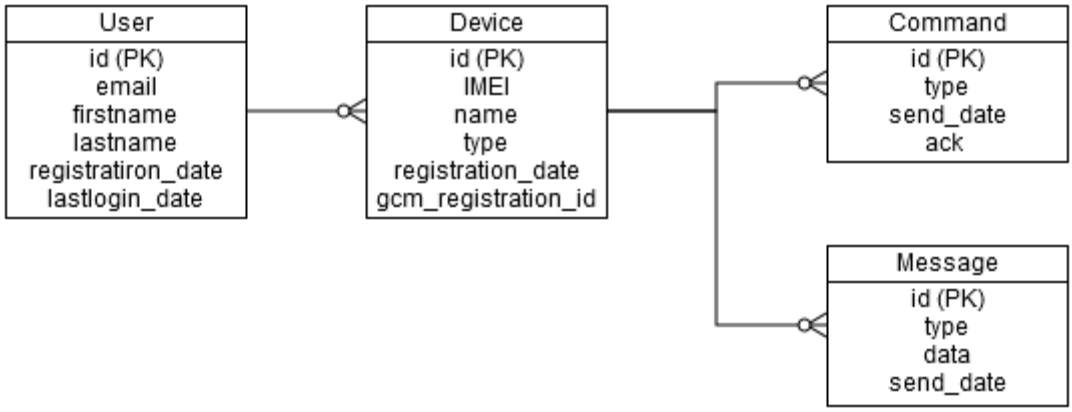
na stejném Google účtu. Drátový model na obrázku Obrázek **1** ukazuje, jak bude webová aplikace vypadat po přihlášení. Tento grafický návrh vyplývá z předchozího diagramu použití.

Jsou zde vidět tlačítka pro vyvolávání jednotlivých akcí definovaných v kapitole 5.1. V pravé části obrazovky je panel, kde se vypisují data získaná z telefonu. Horní panel slouží pro přepínání mezi uživatelovými telefony a pro přehled, který Google účet je právě přihlášený.

Spodní stavová lišta poté slouží pro zobrazení akcí, které právě probíhají na pozadí. Celá webová aplikace má na pozadí interaktivní mapu, na které se bude zobrazovat buď aktuální poloha PC s otevřeným prohlížečem, nebo v případě lokalizace telefonu, umístění tohoto telefonu.

5.2.2 Návrh databázové struktury

Webová aplikace má jen pár entit. Ty jsou zobrazeny na následující ER diagramu. Každý uživatel aplikace může mít 0 až mnoho zařízení (např. tablet, osobní telefon, služební telefon, a jiné). Na každé zařízení se posílají příkazy. A každé zařízení zpět posílá zprávy. Ty nutně nemusí záviset na žádném příkazu.



Obrázek 16 ER diagram

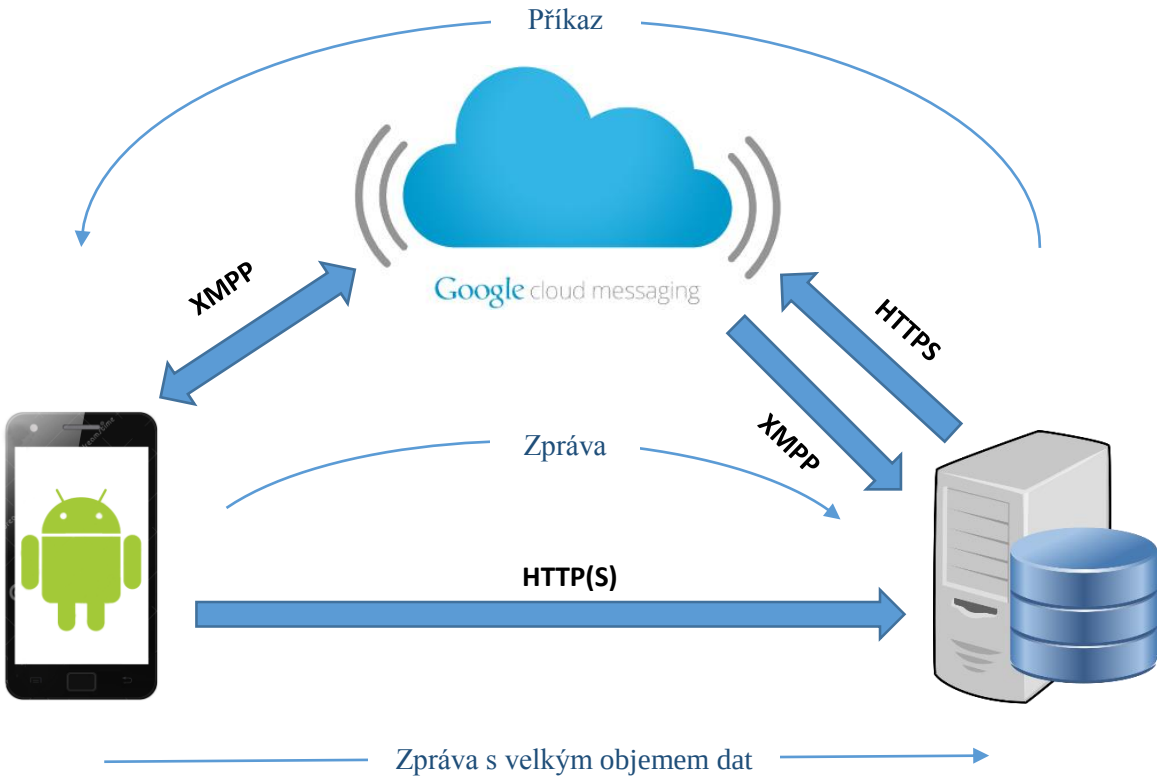
Ve zprávách chodí data z telefonu i v případě, že se například změnila SIM karta, nebo když zpráva o tom, že někdo špatně zadal PIN a podobně. Jednotlivé druhy zpráv budou později rozepsány v kapitole Implementace.

5.2.3 Návrh komunikace s mobilní aplikací

Komunikace mezi webovým serverem a mobilní aplikací bude probíhat podle schématu na obrázku Obrázek **17**. Po zavolání libovolné akce pošle server daný *příkaz* do mobilního telefonu. Směrem od

serveru k mobilnímu telefonu se data (příkazy) posílají přes server Google Cloud Messaging. Ten zajistí, že se zpráva na telefon dostane i v případě, že je právě teď vypnutý. Webový server pošle data na GCM server pomocí HTTPS protokolu. Tento způsob je popisován v kapitole 3.3.1. Mobilní aplikace s GCM serverem komunikuje přes XMPP (CSS) protokol pomocí Android SDK knihoven.

Odpovědi na příkazy zasílá mobilní aplikace opět pomocí GCM přes XMPP (CSS) protokol na webový server, kde běží skript udržující spojení s XMPP serverem, který příchozí zprávy ukládá do databáze. Pokud je zpráva větší než 4kB [2]je z telefonu zaslána pomocí http nebo https POST požadavku. Zasílání zpráv na server pomocí GCM zajistí, že v případě kdy je telefon off-line, pošle zprávy hned po připojení k internetu. U zpráv zasílaných přes http POST bude nutné si to zajistit sám.



Obrázek 17 Schéma komunikace Webového serveru a mobilní aplikace

[2] 4kB jsou maximální velikost zprávy posílané na GCM server

5.3 Návrh mobilní aplikace

V této kapitole bude rozebrán návrh mobilní aplikace na platformě android. Tato aplikace bude přijímat *příkazy* z webového rozhraní. Příkazy budou přicházet z Google Cloud Messaging služby a aplikace na ně bude reagovat. Následující kapitola popisuje jednotlivé obrazovky aplikace.

5.3.1 Grafické návrhy jednotlivých obrazovek

Mobilní aplikace se skládá z několika obrazovek (v Android terminologii z několika Activity). Návrhy jednotlivých obrazovek jsou vidět na následujících obrázcích. První obrazovka je pro funkci *prozvonit telefon*. Tato obrazovka se na telefonu objeví, v případě že majitel telefonu ve webovém rozhraní stiskne příslušné tlačítko. Pozadí této obrazovky bude blikat několika barvami, aby se telefon například ve tmě dal dobře nalézt. Ze stejného důvodu pude blikat i zadní LED dioda sloužící standardně jako blesk k fotoaparátu. Kromě toho bude při této obrazovce telefon vibrovat a vyzvánět sirénou. Po stisknutí tlačítka „Mám tě!“ se obrazovka i všechny zvukové a další efekty vypnou. V případě že je telefon i uzamčen zobrazí se poté následující obrazovka.

Obrázek 18 Wireframe obrazovky prozvánění telefonu a obrazovky uzamknutého telefonu

Další je obrazovka uzamčeného telefonu. Tato obrazovka se objeví při zadání volby uzamknout telefon z webového rozhraní, nebo při aktivaci módu krádeže přes SMS (V tomto případě ještě začne telefon spouštět zvuk říkající „Tento telefon byl právě ukraden“. Přibližně ve středu je umístěno tlačítko *„Zavolat majiteli“*. Po kliknutí na něj se začne vytáčet zadané telefonní číslo. Dále následuje prostor

pro zprávu, kterou může majitel telefonu z webového rozhraní napsat. Ta se dá kdykoliv přepsat na nějakou jinou. Úplně dole je vidět pole pro zadání PINu. To je jediná možnost jak tuto obrazovku vypnout. Dokud se nezadá správný PIN tak se do telefonu nikdo nedostane. Při každém špatném zadání PINu se ještě na přední kameru vyfotí člověk, který má telefon aktuálně v ruce a fotografie se pošle na server.
Tato obrazovka naskočí i při opětovném nastartování telefonu po restartu.

Aplikace bude mít ještě jednu jednoduchou obrazovku, kterou uživatel uvidí pravděpodobně jen jednou, a to při instalaci aplikace. Bude zde především tlačítko pro povolení Administrátorských práv k telefonu a tlačítko pro registraci zařízení do GCM a spárování s webovým rozhraním.

5.3.2 Naslouchání systémových událostí

V této kapitole se zaměřím na návrh tříd, které budou mít za úkol naslouchat na systémové události. Jsou to již dříve zmiňované Broadcast recivery. V mojí aplikaci jich bude muset být několik.

Hlavní bude přijímání příkazů z webové aplikace přes Google Cloud Messaging. K tomu bude sloužit `GcmBroadcatReciever`, který odchytí příchod zprávy a předá ji ke zpracování. Dále bude potřeba odchytit událost, kdy telefon přejde do uzamknutého stavu, kvůli otevření mojí vlastní obrazovky uzamknutí. Ten se bude nazývat `LockScreenReciver`. Jeho další funkcí bude ještě odchytnutí události, že zařízení bylo spuštěno a v případě, kdy je telefon uzamknutý rovnou spustí tutéž obrazovku.

Pak zde musí být také naslouchání na události zadání špatného hesla, či úspěšného odemčení zařízení. K tomu bude sloužit `DevicePolicyReciver`. Pokud jedna z těchto událostí nastane, zašle se příslušná zpráva do zařízení. Pro reagování na opětovné připojení k internetu bude sloužit třída `NetworkChangeReciver` . Ten v případě, že se zařízení přejde do stavu online zkontroluje jestli je potřeba poslat nějaké zprávy s objemnými daty, které nelze poslat přes GCM a pošle je po HTTP.

Pro reagování na příchozí SMS zprávy bude sloužit `SmsCommandReciver`. Ten po přijetí SMS zprávy vyhodnotí, jestli jde o SMS s příkazem, a pokud ano příkaz předá ke zpracování. A poslední broadcast reciever v mojí mobilní aplikaci bude `SimChangedReciver`, který reaguje na zprávu o tom, že byl změněn stav SIM karty. Například když byla vyjmuta, nebo když byla vložena nová. Všechny zjistitelné údaje o stavu zařízení a SIM karty poté pošle zprávou do webové aplikace.

6 Implementace

Kapitola implementace rozebírá realizační detaily praktické části diplomové práce. Je zde rozebrána zvlášť mobilní aplikace a zvlášť webová, protože každá z nich funguje na jiné platformě. Dále je zde podrobněji popsán tok dat mezi mobilní aplikací a webovým serverem. Uvádím zde hlavní a nejdůležitější části obou aplikací od abstraktního pohledu až po konkrétní třídy v PHP či Javě.

6.1 Mobilní aplikace

Mobilní aplikace je psaná v jazyce Java a využívá Android SDK s API verze 16 [14]. To znamená, že aplikace bude fungovat na zařízeních s operačním systémem Android 4.1 Jelly Bean a vyššími. Tato verze přináší hodně změn oproti starším verzím a také starší verze mají výrazně menší zastoupení na trhu [8][15]. Proto jsem se rozhodl starší verze Android nepodporovat.

Android SDK používá pro sestavování aplikace automatizační nástroj Gradle. Ten slouží jednak ke stažení závislostí (externích knihoven, které projekt využívá), ale také k sestavení Android aplikace pro spuštění na zařízení či v Emulátoru. Gradle vychází z jeho předchůdců Ant a Maven, známých z vývoje Java aplikací, a také je s nimi zpětně kompatibilní. [16]

6.1.1 Struktura aplikace

Celá aplikace je umístěna v balíčku `cz.vutbr.fit.stud.xslade12.lostphone` a jejích pod balíčkách, kde jsou umístěné tematicky podobné třídy. Jsou to následující balíčky:

- `activities` – třídy pro obsluhu jednotlivých obrazovek aplikace,
- `recievers` – třídy s Broadcast recievery naslouchajícími systémové události (jako například příchozí zpráva přes GCM, změna stavu SIM karty, změna stavu připojení k síti, příchozí SMS či třeba Odemknutí / Zamknutí zařízení),
- `services` – třídy se službami na pozadí,
- `commands` – třídy představující jednotlivé příkazy ze serveru pro telefon,
- `messages` – třídy představující jednotlivé typy zpráv z telefonu na server.

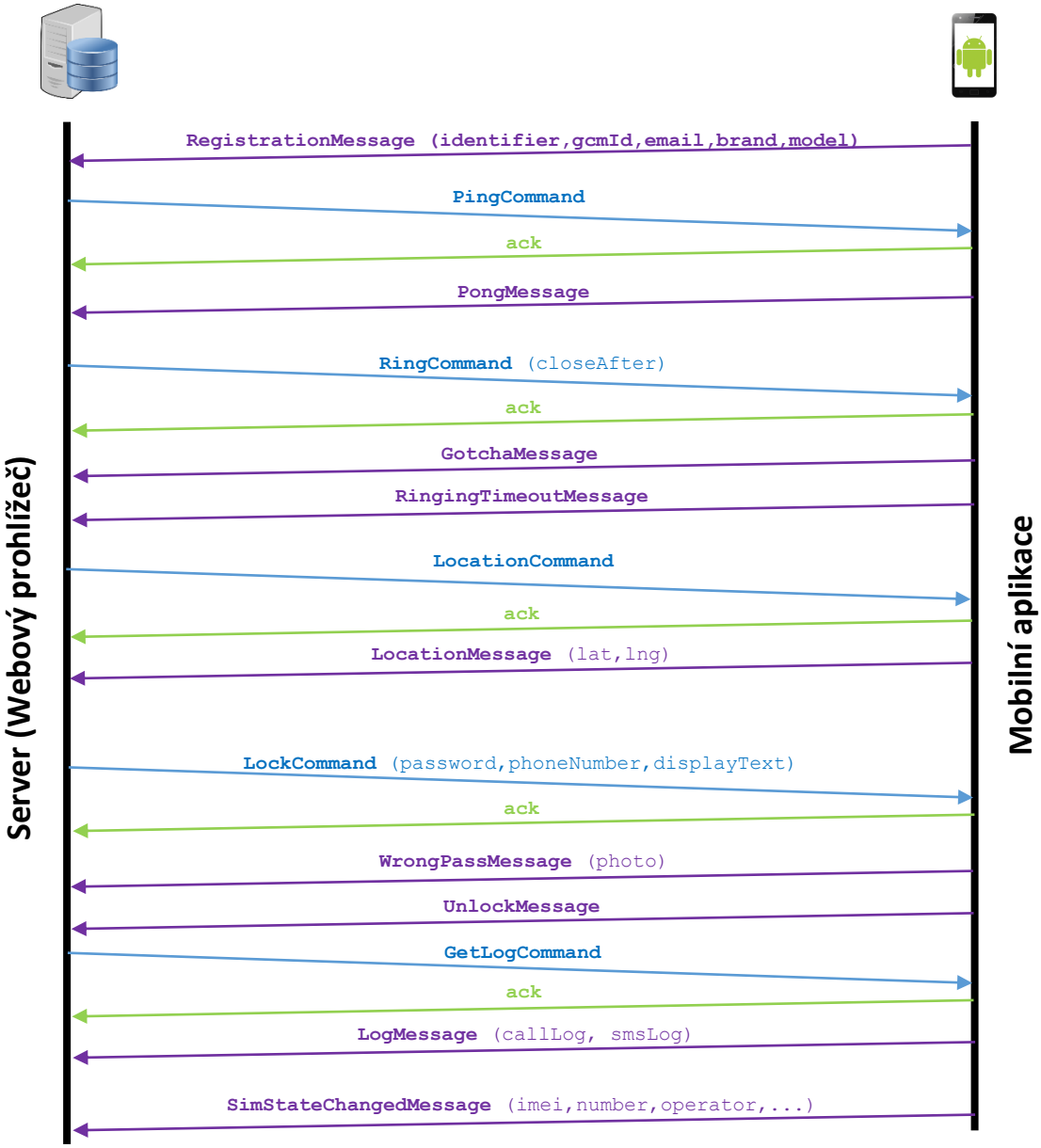
V hlavním balíku je důležitá třída `Worker`, která zajišťuje zpracování příchozího příkazu, a také zasílání zpráv na server. Jejím úkolem jsou také další globální dílčí úkony, jako například čtení a zapisování dat do datového úložiště na zařízení, ke kterému přes tuto třídu ostatní aktivity či *Broadcast Recievery* přistupují.

Dále aplikace obsahuje třídy pro přístup k přední kameře či obstarávající lokalizaci telefonu. O nich bude řeč v následujících kapitolách.

6.1.2 Příkazy a zprávy

Mobilní aplikace funguje na principu přijímání příkazů a zasílání zpráv. Zpráva může přijít jako reakce na příkaz (např. zašli seznam posledních volání) nebo také asynchronně po vyvolání nějaké události v zařízení (např. špatné zadání hesla).

Jednotlivé příkazy a zprávy ukazuje diagramu na Obrázek 19. Všechny zprávy dědí od abstraktní třídy `Message` a všechny příkazy od abstraktní třídy `Command`. Každý příkaz i zpráva mohou obsahovat vlastní parametry. V diagramu jsou zapsány v závorce za názvem příkazu či zprávy. Po doručení příkazu ze serveru na zařízení odešle zařízení potvrzující ACK zprávu s ID příkazu, která uživateli webové aplikace oznámí, že příkaz byl do zařízení doručen, i v případě, že ještě nedošla žádná zpráva s užitečnými daty (např. Lokalizace telefonu může chvíli trvat).



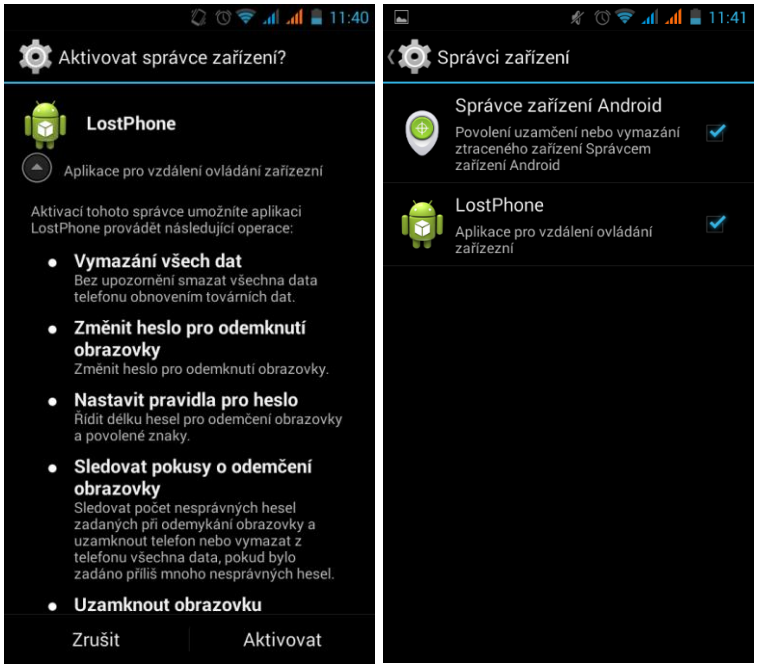
Obrázek 19 Diagram příkazů a zpráv

Velký problém při zasílání zpráv, ze zařízení na server je ovšem jejich velikost. Google Cloud Messaging dovoluje posílat zprávy s velikostí dat do max. 4kB v jedné zprávě a pro větší data je doporučenou použít jinou cestu. V našem případě se jedná o zprávu obsahující fotografii z přední kamery. Ta může mít i několik desítek MB. Řešení se nabízí několik:

a) Posílat velká data přes HTTP POST přímo na server a rovnou je zaspat na disk či do databáze,
b) posílat data na e-mail, který bude server průběžně kontrolovat přes IMAP a nové zprávy stahovat a zapisovat na disk či do databáze,
c) nahrávat data na FTP server,
d) případně se přes sockety připojit na vlastní službu přijímající větší data.

Každé z těchto řešení má své klady i zápory. Ve většině případů je problém, když je telefon momentálně off-line v čas kdy má zaslat tuto zprávu. Pro malé zprávy to GCM řeší, a po připojení k síti všechny neodeslané zprávy odešle, ale v tomto případě musíme frontu neodeslaných zpráv řešit sami a po připojení k síti všechny neodeslané zprávy poslat.

V mojí aplikace jsem zvolil moznost a) a fotografie posílám pomocí HTTP POST požadavku přímo na webový server. Binární data jsou v tomto případě kódovaná do *multipart/form-data*, stejně jako když se z webové stránky odesílá formulář obsahující soubory. V případě že je zařízení offline se fotografie uloží do složky, a jakmile se telefon opět připojí k síti všechny soubory z této složky se odešlou a poté smažou.



Obrázek 20 Aktivace a deaktivace správce zařízení

6.1.3 Správce zařízení

Aby aplikace měla přístup ke kritičtějším funkcím zařízení, musí být schválena uživatelem jako *Správce zařízení*. To bohužel nestačí jen přidat do seznamu oprávnění, které se schvalují při instalaci aplikace, ale je potřeba při prvním spuštění aplikace tuto aplikace zaregistrovat jako správce zařízení, a potvrdit všechny práva pro správce zařízení (viz Obrázek 20 vlevo).. Správci zařízení jsou k nalezení v nastavení zařízení v pod položkou *Zabezpečení* / *Správci zařízení* a také se zde dají zrušit (viz Obrázek 20 vpravo).

Práva pro povolení jsou zapsána v XML souboru ve složce *device_policies.xml* ve složce *src/main/res/xml/* (tzn. je součástí tzv. resources). Obsah tohoto souboru je vidět na Obrázek 21 a jednotlivé oprávnění znamenají:

- Nastavit pravidla pro heslo.
- Sledovat pokusy o odemčení obrazovky.
- Změnit heslo pro odemknutí obrazovky.
- Uzamknout obrazovku.
- Vymazání všech dat.
- Nastavení vypršení hesla obrazovky.
- Nastavit šifrování úložiště.
- Vypnout fotoaparáty.
- Zakázat funkce v zámku zařízení.

```
<device-admin xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-policies>
        <limit-password />
        <watch-login />
        <reset-password />
        <force-lock />
        <wipe-data />
        <expire-password />
        <encrypted-storage />
        <disable-camera />
        <disable-keyguard-features />
    </uses-policies>
</device-admin>
```

Obrázek 21 XML s právy pro správce zařízení; Zdroj [17]; provedení vlastní

Pro vyvolání okna s potvrzením práv, definovaných v XML souboru stačí zavolat intent s názvem `android.app.action.ADD_DEVICE_ADMIN` a jako extra parametr mu předat jednak název Broadcast recieveru, který bude poslouchat tyto události a také textový extra parametrs

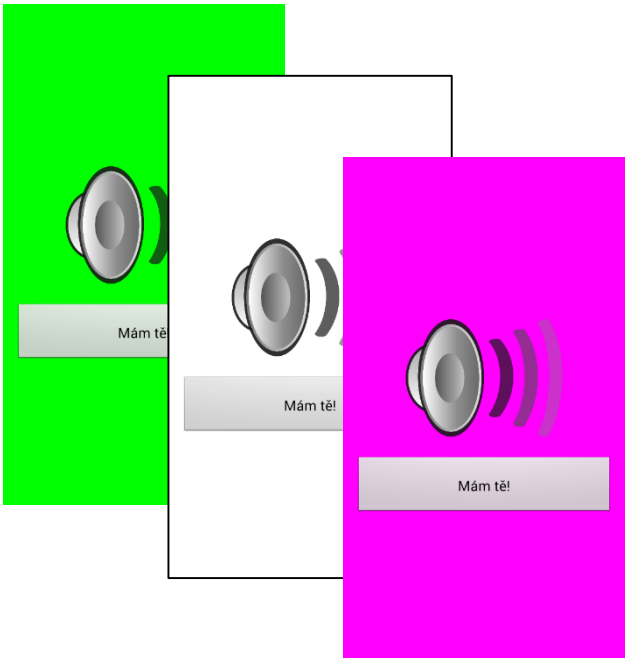
s vysvětlením proč aplikace o tyto práva žádá. Toto všechno se v mojí aplikace řeší v MainActivity po stisknutí tlačítka „Povolit administraci zařízení“. [17]

Vlastní broadcast reciever musí ještě dědit od `DeviceAdminReciever` třídy a v mém případě je implementovaný ve třídě `DevicePolicyReciever`. Jsou zde metody `onEnabled(…)` a `onDisabled(…)` volané pro aktivaci či deaktivaci správce zařízení. Dále pak metody `onPasswordFailed(…)` či `onPasswordSuccessed`(..), na které aplikace naslouchá a zasílá na server zprávy a dále pak jiné pro nás nepodstatné metody. Tento broadcast reciever je samozřejmě uveden v Android Manifestu a jako meta inforamce je k němu přidáno právě XML definující oprávnění pro tuto aplikace (viz výše).

6.1.4 Obrazovka prozvonění

Obrazovka prozvonění je implementována ve třídě `RingingActivity` dědící od abstraktní třídy `WithSoundActivity`. Jejím hlavním úkolem je, aby posloužila k nalezení telefonu / tabletu například někde v bytě. Zajišťuje to třemi základními funkcemi:

1. Zvukovou sirénou,
2. blikáním displeje výraznými barvami + blikáním LED diody sloužící jako blesk k fotoaparátu, v případě, že by zařízení bylo displejem dolů,
3. vibracemi.



Obrázek 22 Obrazovka prozvonění s blikajícím pozadím

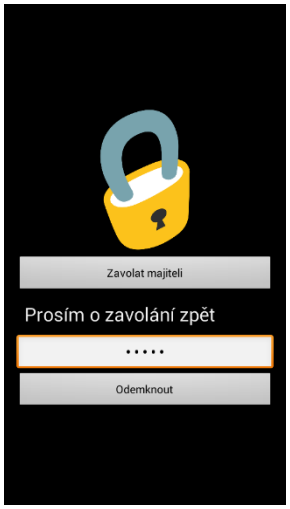
Aktivity dědící od `WithSoundActivity` mají k dispozici metody `soundOn(int resource)` a `soundOff()`. To kromě prostého spuštění zvuku na pozadí zajistí i vypnutí bluetooth,

kvůli případnému odpojední Bluetooth headsetu a také zesílení hlasitosti zařízení na nejvyšší možnou hlasitost. Po zavolání *soundOff*, se vše opět vrátí do původního stavu. Aktivita je má díky nastaveným parametrům okna následující vlastnosti. Je na celou obrazovku (tzv. fullscreen). Po spuštění se zapne osvětlení displeje a po celou dobu běhu se nezhasíná. A v neposlední řadě se aktivita dokáže spustit, i když je telefon zamknutý, přes zamykací obrazovku. To vše má na starost metoda `makeFullScreen()` v `RingingActivity`.

Po zmáčknutí tlačítka „Mám tě!“ se aktivita ukončí a poté zabije svůj vlastní proces, aby nezůstávala spuštěná na pozadí. Dále také zašle zprávu `GotchaMessage` na server. Pokud po uplynutí časového limitu, které lze předat jako parametr closeAfter, na tlačítko „Mám tě!“ nikdo neklikne, aktivita se také ukončí, ale pošle se zpráva `RingingTimeoutMessage` a zároveň se začne telefon lokalizovat, což po chvíli zašle zprávu `LocationMessage` s aktuální polohou telefonu.

6.1.5 Obrazovka uzamknutí

Když uživatel přes webové rozhraní uzamkne zařízení, spustí se tato obrazovka. Jejím hlavním úkolem je, aby se neoprávněný uživatel nemohl dostat do zařízení, ale na druhou stranu aby se mohl lehce ozvat majiteli zařízení v případě, že chce nález nahlásit a zařízení vrátit zpět do rukou majitele.



Obrázek 23 Obrazovka uzamknutí

Programátorské API androidu ovšem nedovoluje udělat Activitu, který by nešla vypnout tlačítkem *Home* a také z přehledu spuštěných aplikací. Stejně tak ve veřejném API není způsob jak vytvořit vlastní *keyguard screen* nahrazující ten standartní s odemykáním slajdem, pinem, heslem či gestem. Nakonec jsem tedy zvolil následující řešení.

Obrazovka uzamknutí se stejně jako obrazovka prozvonění za pomocí `WindowManager.LayoutParams` nastaví jako fullscreen (na celou obrazovku) a také jako okno

které se zobrazí nad klasickou obrazovkou zamknutí. Když je přijat příkaz `LockCommand`, nastaví se zařízení heslo pro odemknutí a zařízení se zamkne. Toto je možné díky tomu, že je aplikace zaregistrována jako správce zařízení.

Po klasickém uzamčení se spustí má `LockScreenActivity`, který se zobrazuje přes standartní uzamykací obrazovku. Také je ovšem zaregistrovaný `LockScreenReciver`, který obstarává spuštění `LockScreenActivity` vždy, když se zobrazuje standartní obrazovka uzamčení (tzn. po rozsvícení displeje, po zapnutí zařízení, po ukončení hovoru atd..)

Vždy když je zadáno špatné heslo pro odemčení, je zaslána fotografie z přední kamery na server. V případě, že je zadáno správné heslo, zruší se uzamčení zařízení a obrazovka zmizí. Při tom je opět zaslána zpráva na server, že bylo zařízení odemčeno.

6.1.6 Ovládání kamery

Přístup ke kameře v zařízení existuje na dvou úrovních. Je možnost pouze vytvořit *Intent* se záměrem vyfotit fotku či nahrát video a předat ho systému. Ten vyvolá uživateli obrazovku, kterou známe například z aplikace Fotoaparát a nechá uživatele fotografii nebo video vytvořit. Poté se vrátí do vaší aplikace a fotografie jako Bitmapa je k dispozici jako parametr.

Druhý způsob je přístup přímo ke kameře za pomocí Framework API. Slouží k tomu objekt `android.hardware.Camera`. V tomto případě je potřeba zjistit identifikační číslo kamery (zařízení jich může mít víc), otevřít kameru, nastavit jí parametry, předat jí View, kde se bude zobrazovat náhled z kamery, vyfotit fotku a uzavřít kameru. Fotografii pak můžeme dostat jako JPEG nebo Bitmap čistá binární data.

K získání ID přední kamery je potřeba nejprve zjistit jestli vůbec zařízení přední kameru má. Na to se dá zeptat manažera balíčků pomocí metody `hasSystemFeature()` s parametrem `PackageManager.FEATURE_CAMERA_FRONT` na objektu `PackageManager`. V případě, že zjistíme, že kamera existuje, musíme projít všechny dostupné kamery a najít, která je přední. Funkce, která toto implementuje je vidět na v následujícím kódu.

```
private int getFrontCameraId(){
      int camId = -1;
      int numberOfCameras = Camera.getNumberOfCameras();
      Camera.CameraInfo ci = new Camera.CameraInfo();

      for(int i = 0;i < numberOfCameras;i++){
            Camera.getCameraInfo(i,ci);
            if(ci.facing == Camera.CameraInfo.CAMERA_FACING_FRONT){
                  camId = i;
            }
      }

      return camId;
}
```

Obrázek 24 Zjištění ID přední kamery

V mé aplikaci využívám druhý způsob, který na rozdíl od prvního nevyžaduje interakci uživatele. Celé ovládání kamery je umístěno ve třídě `FrontCameraController`, která má za úkol pořízení fotografie z předního fotoaparátu na pozadí a bez vědomí uživatele. Dále také pomáhá s ukládáním fotografie do externího úložiště. V kódu na Obrázek 25 je ukázka použití `FrontCameraControlleru`, který bez uživatelova vědomí vyfotí fotografii přední kamerou a uloží ji do souboru `pictureFile`.

```
final FrontCameraController fc = new FrontCameraController(context);
if(fc.hasCamera()) { // Zařízení má přední kameru
    fc.open(); // otevře přední kameru
    fc.setPictureCallback(new Camera.PictureCallback() {
        @Override
        public void onPictureTaken(byte[] data, Camera camera) {
            // Založí novy soubor
            File pictureFile = fc.getOutputMediaFile();

            // Uloží binární data do souboru
            FileOutputStream fos = new FileOutputStream(pictureFile);
            fos.write(data);
            fos.close();
        }
    });
    fc.takePicture();
} else {
    // Zařízení nemá přední kameru
}
```

Obrázek 25 Veřejné API FrontCameraControlleru

6.1.7 Lokace zařízení

Zjištění polohy zařízení lze provádět dvěma způsoby. Nejpřesněji to jde pomocí GPS. Dále pak lze určit polohu triangulací vzdálenosti vysílačů, případně Wi-Fi přístupových bodů. V androidu Framework existuje třída `LocationManager`, která má za úkol zjišťovat polohu zařízení. Jako vstup bere tzv. provider, čili poskytovatel lokace. Jedním z nich je `LocationManager.GPS_PROVIDER` pro zjišťování polohy právě z GPS satelitů a druhým `LocationManager.NETWORK_PROVIDER`, který zjišťuje polohu ze sítě.

Pro mou aplikaci jsem napsal třídu `LocationController`, která má za úkol zjistit polohu alespoň z jednoho z těchto providerů. Pokud se to nepodaří, tak po uplynutí limitu 20 sekund vrátí alespoň poslední známou polohu. Po zjištění polohy zavolá callback a předá mu zjištěné souřadnice.

6.2 Webová aplikace

V této podkapitole budou zmíněny implementační detaily webové aplikace a to jak jejího serverového *backendu*, tak jejího *frontendu* na straně prohlížeče.

Serverová část aplikace je psaná v jazyce PHP 5.5. Využívá několik externích knihoven a pro její seskládání slouží balíčkovací systém Composer. Nejdůležitějšími balíčky jsou Nette Framework ve verzi 2.2.8 a Doctrine ve verzi 2.4.7. Nette zajišťuje MVC strukturu aplikace, šablonový systém Latte, překreslování šablon ajaxem za pomocé tzv. snippetů a další užitečné věci. Doctrine je ORM systém pro PHP a MySQL databázi. Jeho hlavní funkcí je persistovat datové entity do databáze a opět je z ní číst.

Klientská část aplikace, kterou zpracovává webový prohlížeč je psána ve značkovacím jazyce HTML5, prototyp grafiky je udělaný za pomocí CSS frameworku Bootstrap a využívá pro svou funkci Javascriptu.

6.2.1 Struktura aplikace

Složka s aplikací obsahuje 6 hlavních složek. Složka *log/* slouží k ukládání logů (informačních, chybových). Do složky *temp/* se ukládají dočasné soubory a souborové keše, například pro šablony nebo konfiguraci DI kontejneru. Do složky *upload/* se ukládají nahrané soubory (v našem případě fotografie z telefonu) a složka *www/* obsahuje veřejně dostupné soubory pro webový server (kaskádové styly, soubory s javascriptem, obrázky pro grafiku a soubor *index.php*, který spouští celou aplikaci. Dále je zde důležitý adresáš *vendor/*. Do něho se ukládají všechny externí knihovny, na kterých je moje

aplikace závislá. O závislosti se stará balíčkovací systém Composer a všechny závislosti jsou definované v souboru *composer.json* v rootu aplikace.

Zdrojové kódy samotné aplikace jsou umístěny ve složce app/ a jejích podsložkách. Ty se jmenují podle názvu namespace tříd v nich uložených. Ve složce *app/commands/* jsou například třídy sloužící pro spuštění z příkazové řádky. Ve složkách app/*presenters/*, app/*templates/* a *model/* jsou kontroléry (Presentery), šablony a modelové třídy z MVC struktury. Dále ve složce app/*services/* jsou servisní třídy, vykonávající tzv. bussines logiku aplikace. Dále je zde složka s konfiguračními soubory (*app/config/)* a složka s nastavením routování (*app/router/*).

Aplikace se snaží zachovávat osvědčené návrhové vzory. Jednak výše zmiňovaný MVC (Model–view–Controller), pro rozdělení vrsty šablony, datového modelu a logiky aplikace, ale také např. Dependency injection, který říká, že závislosti každé třídy musí být veřejné, a třída si je nemá nikde získávat sama. Využité těchto a jiných návrhových vzorů pomáhá k znovu použitelnosti zdrojových kódů z aplikace.

6.2.2 Datové entity v Doctrine

Už dříve zmiňovaná Doctrine slouží jako ORM pro PHP. Zkratka ORM znamená Object Relation Mapper a stará se o persistování datových entit do databáze a jejich opětovné načítání. Datová entita je prostá třída, která nese užitečná data. Jednotlivé entity jsou uvedeny v návrhu datového modelu v kapitole specifikace. Aby Doctrine věděla, se kterými třídami má pracovat a jak přesně, jsou označeny tzv. anotacemi. Každý entita musí mít nad definicí třídy uvedenou anotaci `@Entity` a případě anotaci `@Table(name="NazevTabulky")` pokud chceme určit název tabulky v databázi. [18]

```
namespace App\Model;

use Doctrine\ORM\Mapping\Entity;
use Doctrine\ORM\Mapping\Table;

/**
 * @Entity
 * @Table(name="users")
 */
class User {
      // ...
}
```

Obrázek 26 Definice Doctrine entity

Kromě anotací nad celou třídou je potřeba ještě uvádět anotace u jednotlivých vlastností této třídy. Tedy u těch, které se budou do databáze persistovat. Je potřeba uvést datový typ vlastnosti,

případně je opět možné uvést odpovídající název sloupce přímo v databázi či označit že vlastnost může být prázdná.

```
// ...
    /** @Column(type=string, nullable=true) */
    protected name;
```

Obrázek 27 Definování sloupců v entitě

S pomocí anotací lze určit i indexy a další upřesnění. Doctrine ovšem jako správné ORM umí také relace 1:1, 1:N či M:N. Slouží k tomu anotace `@oneToOne`, `@oneToMany` a `@manyToMany` s příslušnými parametry upřesňujícími tento vztah. Pomocí anotací lze v Doctrine také ovlivnit persistování entit, které využívají dědičnosti tříd v PHP. [18]

Velká výhoda ORM je také odstínění od databáze. Programátor nemusí řešit návrh databáze, ani není důležité, jaká databáze se použije. Stačí si napsat třídy Entit a z příkazové řádky spustit příkaz, který entity ve složce vyhledá a vytvoří podle nich v databázi správné schéma. Doctrine má také nástroje pro změny schématu případně i s využitím databázových migrací (verzování databáze).

Pro vytahování objektů z databáze slouží speciální jazyk DQL. Na první pohled se velmi podobná klasickému SQL, ovšem pracuje se v něm s entitami a jejich vlastnostmi jako s objekty, ne jako s řádky v SQL. Podobný způsob se používá i v Javě v JDO či v .NETu s Entity Frameworkem.

6.2.3 Třída HTTP Sender

Pro odesílání zpráv na Google Cloud Messaging server přes HTTP protokol slouží třída z `Gcm/Http/Sender`. Je to služba, jejíž konstruktor přijímá API klíč googlu a má jednu veřejnou metodu `send($message)`, která přijímá objekt představující GCM zprávu.

```
use Gcm\Message;
use Gcm\Http\Sender;

$message = new Message("DEVICE_GCM_ID",
                       ['foo'=>'bar', 'baz'=>[1,2,3]],
                       "collapse-key-1");
$message->addTo("ANOTHER_DEVICE_GCM_ID");
$message->timeToLive(3600); // TTL 1 hour

$gcm = new Sender("API_KEY");
$response = $gcm->send($message);

var_dump($response);
```

Obrázek 28 Použití služby Gcm\Http\Sender

Třída `Gcm\Message` slouží jako obálka na data zasílaní přes GCM s objektovým přístupem. V prvním parametru konstruktoru přijímá ID příjemce, v druhém pole s uživatelskými daty a ve třetím tzv. *collapse key*, což je klíč podle kterého se zprávy seskupují v případě že je zařízení příjemce off-line. Třída osahuje také gettery a settery na všechny vlastnosti zprávy definované v Tabulka 3. Pokud je zpráva posílaná přes HTTP je možné ji nastavit 1 až 1000 příjemců. K tomu slouží buď metoda `addTo($deviceGcmId)` nebo třída v prvním parametru konstruktoru přijímá pole s ID příjemců.

Služba Gcm\Sender v metodě `send($message)` správně serializuje data z `Gcm\Message` a pošle je pomocí CURLu HTTP POST požadavkem na server GCM. Připojí správné hlavičky pro autentizaci a také ošetří, jestli má správa příjemce a jestli není moc veliká. Pokud je něco špatněvyhodí příslušnou výjimku. Po odeslání vrátí objekt typu `Gcm\Http\Response`, který obsahuje odpověď od GCM serveru. Je v něm uvedeno kolika příjemcům bude zpráva úspěšně doručena, a kolik je např. neexistujících a případně také obsahuje pole s detailním výpisem pro každého příjemce.

Této služby se využívá při zasílání příkazu z webového rozhraní pro kliknutí na tlačítko (odkaz) či odeslání formuláře.

6.2.4 Třída XMPP Daemon

Protože přes GCM po HTTP lze zasílat zprávy jen ze serveru na zařízení, pro příjem zpráv je potřeba využít spojení přes protokol XMPP. Pro tento případ jsem napsal třídu `Gcm\Xmpp\Daemon`, která obstarává připojení ke GCM CCS serveru (Cloud Connection Server) právě pomocí XMPP protokolu a zapouzdřování zpráv do správného JSON payloadu, který Google vyžaduje.

XMPP (Extensible Messaging and Presence Protocol) je protokol, který se používá k rychlému zasílání zpráv (tzv. Instant messagingu), jako je například Jabber nebo Facebook chat. Funguje přibližně tak, že se pomocí socketů připojí na daný server na příslušném portu, a pomocí XML zpráv se s ním komunikuje. V PHP existuje knihovna `JAXL`, která slouží právě pro připojování na Jabber chat, ale pro komunikaci s CCS serverm Googlu, bylo potřeba ji trochu upravit.

Moje třída `Gcm\Xmpp\Daemon` má závislost právě na knihovně `JAXL`, kterou využívá pro zasílání a přijímání zpráv. Potřebné úpravy jako je například změna protokolu z *tcp* na *ssl* nebo změna způsobu autentifikace je provedena pomocí dědičnosti a přetížení některých metod. Daemon také řeší implementaci kontroly toku za pomocí ACK a NACK zpráv a nabízí uživateli veřejné rozhraní odstíněné od těchto problémů.

Příklad použití je vidět na Obrázek 29. Do konstruktoru třídy Daemon se předná ID odesílatele a API klíč, který jsme získali při registraci aplikace v administrátorské konzoli Googlu na adrese https://console.developers.google.com. Dále se na objektu definují callbacky pro jednotlivé události. Jednotlivé události by měli být zřejmé z ukázky kódu. Callbacky se přidávají do pole, protože jich může být více na jednu událost. Například příjde-li přes sockety zpráva z GCM serveru zavolají se všechny události v poli `onMessage`.

```
use Gcm\Xmpp\Daemon, Gcm\Message, Gcm\RecievedMessage;

$daemon = new Daemon("SENDER_ID", "API_KEY", $testMode = false);
$daemon->onReady[] = function(Daemon $daemon) {
    print "Připraven, prihlášení proběhlo úspěšně";
    foreach([1,2,3,4,5] as $i) { // pošleme 5 zpráv
        $msg = new Message("DEVICE_GCM_ID",
                           ['text'=>"$i.zpráva"],
                           "collapse-key-$i");
         $daemon->send($msg); // odešli
    }
};
$daemon->onAuthFailure[] = function(Daemon $daemon, $reason) {
    print "Přihlášení nebylo úspěšné (z důvodu: $reason)";
};
$daemon->onStop[] = function(Daemon $daemon) {
    print 'Spojení bylo zastaveno příkazem $daemon->stop()';
};
$daemon->onDisconnect[] = function(Daemon $daemon) {
    print "Spojení bylo přerušeno";
};
$daemon->onMessage[] = function(Daemon $daemon, RecievedMessage $msg) {
    print "Přišla nova zpráva z GCM";
    print_r($msg);
};
$daemon->onAllSent[] = function(Daemon $daemon, $countMessages) {
    print "Bylo odesláno $countMessages zpáv";
    // Všechny odeslané zprávy byli úspěšně potvrzené
    $deamom->stop(); // Přerušíme spojení
};
$daemon->run(); // Spustit (poběží do zavolání $daemon->stop)
```

Obrázek 29 Příklad použití Gcm/Xmpp/Daemon

V ukázce je vidět že po připojení k CCS serveru se odešle 5 zpráv a jakmile jsou všechny úspěšně potvrzeny, spojení se ukončí a skript skončí. V případě, že mezitím byla přijata jiná zpráva od zařízení, vypíše se na standartní výstup. Objekt zprávy pro poslání je stejné jako v předchozí kapitole. Ovšem pro zasílání přes XMPP smí zpráva obsahovat jen jediného příjemce. Více příjemců je potřeba rozdělit do více zpráv. Příchozí zprávy představuje objekt typu `Gcm\RecievedMessage`, který disponuje

metodami pro získání příchozích dat v poli `getData()`, zjištění GCM ID příjemce `getFrom()` apod.

Hlavní problém pro moji aplikaci, je že je nutné udržet připojení a naslouchat příchozí zprávy pořád, bez odpojení. Proto je v aplikaci vytvořený konzolový příkaz, který musí běžet na serveru na pozadí, a vždy když přijde z libovolného zařízení zpráva, uloží ji databáze, odkud je poté přečtena a zobrazena na webu. Tento příkaz se spouští z příkazem `php index.php app:daemon -v`, kde *app:daemon* určuje, který příkaz v aplikaci se má pustit a parametr *–v* říká že má být „výřečný“ a vypisovat stavové informace na standartní výstup.

Knihovna pro naslouchání zpráv přes XMPP nebyla nikde volně dostupná a proto jsem se rozhodl celou knihovnu pro GCM (tzv. HTTP Sender i XMPP Daemon) zveřejnit jako OpenSource. Git repozitář je i jednoduchou dokumentací je veřejně dostupný na GitHubu na adrese https://github.com/PetrSladek/Gcm a také jako balík pro balíčkovací systém composer pod názvem *petrsladek/gcm* na adrese https://packagist.org/packages/petrsladek/gcm. Nainstalovat do projektu lze jednoduše pomocí následujícího příkazu.

```
composer require petrsladek/gcm:dev-master
```

Také do mého projektu je připojena jako závislost, která se automaticky stahuje při instalaci composer a proto jsou její zdrojové kódy na přiloženém CD zvlášť od zdrojových kódů webové aplikace.

6.2.5 Přenášení událostí ze serveru do prohlížeče

Ve chvíli kdy dorazí zpráva z mobilního zařízení na XMPP daemona je také potřeba zobrazit tuto zprávu uživateli ve webovém prohlížeči. Vzhledem k tomu, že HTTP protokol, kterým webový prohlížeč stahuje webovou stránku ze serveru, funguje na principu požadavek – odpověd (request – response), neexistuje možnost, jak by http server dokázal stránku při přijetí zprávy aktualizovat. Nabízí se teda několik možností jak klientovy dát o nových zprávách vědět:

1. Periodicky po několika vteřinách aktualizovat stránku,
2. využít Server-Send Events API,
3. nebo využít Web Socket API.

Periodická aktualizace celé stránky rozhodně není uživatelsky příjemná, ale dá se pomocí AJAXu aktualizovat pouze určitá část stránky, v našem případě ta část, kde se vypisují zprávy z telefonu. Tato možnost je v aplikaci implementována, protože funguje i na starších prohlížečích a také se díky tomu jedno začas ukáží i zprávy, které nepřišli kvůli velikosti přes GCM, ale vložili se do databáze přes POST požadavek na server. Nahrazování pouze nějaké části webové stránky má na

starosti Nette Framework pomocí tzv. snippetů. Jsou to definované bloky v šabloně, a při ajaxovém požadavku na server se přenáší jen tyto části šablony zakódované do JSON payloadu. Ten poté klientský javascript zpracuje a aktualizuje DOM strukturu dokumentu.

Server-Sent Events fungují na podobném principu, jen je jejich podpora integrovaná přímo do prohlížeče a udržuje se delší HTTP spojení. Klientská javascript v prohlížeči zaregistruje naslouchač událostí (EventListener) na konkrétní URL a začne naslouchat. Ukázka zdrojového kódu je na Obrázek 30. [19]

```
var messages = new EventSource("http://www.example.com/messages.php");
messages.onmessage = function (event) {
    console.log(event.data)
};
```

Obrázek 30 užití Server-Sent Events v Javascriptu

Soubor *messages.php* v tomto případě musí zasílat HTTP odpověď s obsahem, kde každý řádek začíná znaky „*data:* “ a mezi jednotlivými událostmi musí být prázdný řádek. V tomto případě není žádoucí, aby skript doběhl a odeslal celou odpověď. Využívá se tak zvaný „write buffer“, kam se ukládá vše co se má zaslat zpět klientovi. V PHP se dá funkcí *flush()* odeslat i když stále není ukončený přenos. Tím se udržuje spojení mezi klientským javascriptem a serverem a vždy když příjdou nová data, prohlížeč zavolá `onmessage` funkci na příslušném posluchači. Webový prohlížeč ovšem požadavek po nějakém čase ukončí, poté klient chvíli počká a znovu vytvoří připojení. Ukázka zdrojového kódu v php je na následujícím obrázku. [19]

```
header('Content-Type: text/event-stream');
header('Cache-Control: no-cache');
function sendMessage($msg) {
    echo "data: $msg" . PHP_EOL;
    echo PHP_EOL;
    flush();
}
while(true) {
      $serverTime = time();
      sendMsg($serverTime, 'server time: ' . date("h:i:s", time()));
      sleep(1); // počkej 1 vteřinu
}
```

Obrázek 31 Zasílání Server-Sent Events v PHP

Nevýhoda tohoto řešení je, že se PHP musí v nekonečné smyčce dotazovat MySQL serveru jestli v něm není nová zpráva a až pokud ano, tak ji poslat. Samozřejmě se dá využít aktivního čekání (php funkce `sleep(int seconds)`), ale v tom případě to degraduje na úroveň předchozího řešení s obnovování Ajaxem. Server-Sent Events také nemají vůbec podporu v aktuální verzi Internet Explorer 10. Proto jsem jej v aplikaci nakonec nepoužil a rozhodl jsem se využít poslední zmiňované řešení.

Webové sokety (Web Sockets) slouží právě k obousměrné komunikaci mezi webovým prohlížečem a serverem v reálném čase. Je to obdobná technologie jako klasické sokety, kde server naslouchá na nějakém portu a klient se k němu pomocí jeho IP adresy a čísla portu připojí. Pro připojení k web soketovému servru slouží v javascriptu třída `Websocket`. Do konstruktoru se předává IP adresa nebo doménové jméno serveru s protokolem *ws://* (websocket) nebo *wss://* (websocket secure). Objekt této třídy nabízí tři události s názvem `onopen`, `onmessage` a `onclose`. První z nich se volá po připojení k serveru, druhá příjde-li od serveru nějaké zpráva a třetí po ukončení spojení. Dále objekt disponuje metodou `send()` pro zaslání dat na server [20]. Jako data se zde posílají jen textové řetězce. Složitejší objekty je potřeba serializovat např. pomocí JSONu. Příklad použití je vidět v následující ukázce.

```
if ("WebSocket" in window) { // otestujeme podporu prohlížeče
    var ws = new WebSocket("ws://www.example.com:8889");

    ws.onopen = function(e) {
        console.log("Spojení otevřeno");
        ws.send("Zdravím světe"); // posílám zprávu serveru
    };
    ws.onclose = function(e) { console.log("Spojení ukončeno"); };
    ws.onmessage = function(e) {
        console.log("Přijata zpráva: " + e.data);
    };
}
```

Obrázek 32 Použití WebSocket API

Na straně serveru je potřeba spustit proces, který na příslušném portu naslouchá a reaguje na zprávy. V PHP k tomu slouží funkce `stream_socket_server` pro spuštění serveru, `stream_select` pro konkurenční obsluhování klientů a `stream_socker_accept` pro čtení dat od klienta.

Bylo by ovšem zapotřebí vyřešit všechny náležitosti přesnostu po protokolu WebSocket definovaném v RFC6455 [21] a proto jsem radši zvolil využít externí knihovnu

`Hoa\Websocket\Server` z projektu Hoa[3]. Její instalace pomocí composeru je jednoduchá stačí v příkazové řádce v root adresáři projektu spustit již dřív zmíněný příkaz `composer require hoa/websocket:~2.0` a knihovnu začít používat. Ukázka použití je vidět v následujícím zdrojovém kódu. [22]

```
use Hoa\Websocket\Server;
use Hoa\Socket\Server as SocketServer;
use Hoa\Core\Event\Bucket;

// Server naslouchá na portu 8889;
$server = new Server( new SocketServer('tcp://127.0.0.1:8889') );
$server->on('open', function (Bucket $bucket) {
    echo "Nový klient\n ";
    return;
});
$server->on('message', function (Bucket $bucket) {
    $data = $bucket->getData();
    echo "Nová zpráva " . $data['message'] . "\n";
    // Zprávu pošleme zpět klientovi
    $bucket->getSource()->send("Přišl: " . $data['message']);
    return;
});
$server->on('close', function (Bucket $bucket ) {
    echo "Klient se odpojil\n";
    return;
});
```

Obrázek 33 Použití Hoa\WebSocket\Server v PHP; Zdroj [22]; provedení vlastní

V mojí aplikaci jsou Websockety použity pouze pro upozornění, že jsou v databázi nová data. Samotné snippety s HTML kódem jsou nakonec staženy AJAXem. Celé workflow funguje následovně:

1. Příkazem `php www/index.php app:server` se spustí WebSocketový server, který běží na serveru v pozadí stejně jako dříve zmiňovaný GCM Deaemon.
2. Pokud má uživatel spuštěnou webovou aplikaci, webový prohlížeč se přes WebSockets připojí k tomuto serveru a řekne, že přijímá zprávy pro zvolené uživatelovo zařízení. To

[3] Hoa je modulární, rozšiřitelná a strukturovaná sada PHP knihoven. Webová stránka projektu je http://hoa-project.net/

znamená, že pošle serveru zprávu s ID zařízení, pro které chce přijímat nové zprávy. Server pak drží v paměti, že toto zařízení patří tomuto připojenému klientovi.

3. Když přijde přes z GCM přes XMPP zpráva tak GCM Daemon se připojí jako klient k tomuto serveru a zašle na něj zprávu o tom, že je v databázi pro zařízení identifikované svým ID nová zpráva. To znamená, že pošle serveru zprávu s ID zařízení, pro které nová zpráva přišla.
4. Server zprávu přijme, podívá se, jestli existuje klient, který přijímá zprávy pro toto zařízení, a pokud ano pošle tomuto klientovi zprávu, že jsou pro něj v databázi nová data.
5. Webový prohlížeč přijme od serveru tuto zprávu a AJAXovým požadavkem požádá http server o nový HTML kód (snippet) pro vykreslení. Tím se data stáhnou z databáze, vykreslí do HTML a zobrazí v prohlížeči.

Jistě by se dalo udělat, že snippet s HTML kódem přijde rovnou přes webové sokety, ale mnou navržené řešení by mělo být dostačující. Pokud má klient starší prohlížeč nepodporující WebSockets, tak se nic neděle, akorát mu zprávy budou chodit pomaleji, protože bude muset čekat, až prohlížeč po uplynutí časového intervalu teprve stáhne sám nová data.

```
parameters:
    websockets: # nastaveni websocketu
        serverUrl: tcp://127.0.0.1:8889 # URL kde server naslouchá
        clientUrl: ws://www.lostphone.cz:8889 # veřejné URL k připojeni
    database: # nastaveni připojení k Databázi
        user: username
        password: password
        dbname: database_name
    gcm: # nastaveni Google Cloud Messaging
        apiKey: "<GCM_API_KEY>"
        senderId: "<GCM_SENDER_ID>"
    google: # nastaveni Google API (pro login atd.)
        clientId:"<CLIENT_ID>.apps.googleusercontent.com"
        clientSecret: "<CLIENT_SECRET>"
```

Obrázek 34 Konfigurace webové aplikace

6.2.6 Instalace a spuštění

Spuštění webové aplikace bohužel není jen její nahrání na webový server. Musí se udělat několik úkonů. Prvním z nich je vytvořit pro aplikaci MySQL (případně MariaDB) databázi a importovat do ní soubor *lostphone.sql*, který je umístěn v adresáři s projektem.

```
mysql -u <username> -p -h <host> <database_name> < lostphone.sql
```

Dále je potřeba aplikaci správně nakonfigurovat Konfigurace se nachází v souborech *config.neon* a *config.local.neon* ve složce *config/*. První zmiňovaný soubor obsahuje konfiguraci aplikace, do které není vhodné zasahovat. Jsou v ní nastavení služeb, velikosti obrázků, konzolových příkazů a podobně. Pro spuštění aplikace je nutné zapsat potřebné údaje do druhého souboru **config.local.neon**. Jeho obsah je vidět ve zdrojovém kódu na Obrázek 34 Konfigurace webové aplikace. Je potřeba do něj doplnit údaje pro připojení k databázi, API klíče získané při registraci služeb v Google Developer Console, a také adresy a porty pro nastavení serveru a klienta webových soketů.

Webová aplikace potřebuje práva pro zápis souborů do složek pro logování, dočasné soubory a upload. V Unixových operačních systémech stačí spustit následující příkaz:

```
chmod –R 777 log/ & chmod –R 777 temp / & chmod –R 777 upload/
```

Případně stačí jen nastavit práva pro zápis uživateli, pod kterým běží HTTP server. Poté je potřeba stáhnout všechny externí knihovny, které má projekt uveden jako své závislosti. O to se postará balíčkovací systém composer a následujícím příkazem vše potřebné stáhne a vloží do aplikace.

```
php composer.phar update
```

Posledním krokem je spuštění skriptů co musí bežet po celou dobu na pozadí a naslouchat na příslušných portech. Jsou to již dříve zmiňovaný XMPP klient aneb GCM Daemon a také WebSocket server pro přeposílání informací o nových zprávách do prohlížeče. Oba se spustí následujícími příkazy z příkazové řádky s aktuálním umístěním v adresáři projektu.

```
php www/index.php app:daemon -v
php www/index.php app:server –v
```

7 Testování

Tato část práce se bude věnovat popisu testování mobilní i webové aplikace navržené a implementované podle předchozích kapitol. Cílem je otestovat funkčnost aplikace a její použitelnost v reálných případech dle scénářů definovaných ve specifikaci požadavků na aplikaci. Dále také otestovaní kompatibility aplikace s různými typy Android zařízení. A v neposlední řadě jde také o zjištění případných chyb a jejich odladění před odevzdáním finální verze aplikace.

7.1 Testování kompatibility

Cílem testování kompatibility je ověření správného provozu a funkčnosti aplikace na různých zařízeních a to jak typem, tak verzí operačního systému Android.

Pro testování aplikace jsem zvolil jeden tablet a tři mobilní telefony. Každý z nich má verzi Androidu jinou a také má jiné rozlišení a velikost displeje. Přehled všech testovacích zařízení je na následující tabulce.

Název zařízení	**Verze OS**	**Úhlopříčka a rozlišení**
Lenovo IdeaTab A1000L-F	4.1.2 (Jelly Bean)	7“, 1024x600
Lenovo S750	4.2.2 (Jelly Bean)	4,5“, 960x540
Samsung Galaxy Nexus (i9250)	4.3 (Jelly Bean)	4,65“, 1280x720
LG Nexus 5	4.4 (KitKat)	4,95“, 1920x1080

Tabulka 4 Seznam testovacích mobilních zařízení

Primárním testovacím zařízením, na kterém byla aplikace vyvíjena a průběžně testována pomocí režimu ladění přes USB rozhraní, byl Samsung Galaxy Nexus. Na ostatních zařízení byla aplikace odzkoušena až ve finální fázi a díky nim byli vyřešeny a odladěny chyby, které způsobovali nekompatibilitu či špatné rozvržení jednotlivých obrazovek mobilní aplikace.

Testování webové aplikace probíhalo v nejnovějších verzích obvyklých prohlížečů na operačním systému Windows 7 a Ubuntu 14. Konkrétní verze zachycuje následující tabulka.

Název prohlížeče	**Verze**
Mozzila Firefox	37.0.2
Google Chrome	42.0.23
Microsoft Internet Explorer 11	11.0.96
Apple Inc. Safari	5.0.5

Tabulka 5 Seznam testovacích prohlížečů

Dle otestování na zmíněných zařízeních lze říci, že mobilní aplikace je kompatibilní s zařízeními od Android 4.1 Jelly Bean až po Android 4.4 Kitkat. Letos vydaná nová verze Android 5 Lolipop by měla být zpětně kompatibilní a proto předpokládám, kompatibilitu i s touto verzí. Zobrazení grafického rozhraní jednotlivých obrazovek bylo taktéž v pořádku a díky použití správných layoutů, velikostní jednotky *dp* [4] a také použitím zdrojů (*resources*) pro jednotlivá rozlišení bylo vše správně velké i čitelné na všech testovaných displejích.

7.2 Testování funkčnosti

Účelem testování funkčnosti bylo především ověřit správnou funkci všech operací a ověřit jestli aplikace splňuje všechny požadavky specifikované v kapitole 5.1. Otestovány byly všechny navržené scénáře v reálném prostředí v simulovaných situacích.

Scénář „*Ztracený telefon v bytě“* dle kapitoly 5.1.1 jsem testoval následujícím způsobem postupně se všemi testovacími zařízeními. Telefon/tablet jsem nechal kolegu ukrýt někde v bytě. Poté jsem sednul k počítači a přes Google účet se přihlásil do aplikace. Po vybrání daného zařízení a stisknutí volby „*Prozvonit telefon*“ začal telefon úspěšně zvonit. A to i v případě že měl předtím ztlumený zvuk. Všechny funkce jako vypnutí bluetoooth, vibrace, blikání displeje, ukončení procesu tlačítkem či vypršení a zaslání zprávy po časovém intervalu fungovali taky správně. Proto tento scénář považuji za otestovaný a všechny příslušné funkce za plně funkční.

Dalším scénář je *„Ztracený telefon, který nikdo nenašel“* dle kapitoly 5.1.2. To znamená, že uživatel telefon ztratí někde venku na odlehlém místě, kde ho nejspíš nikdo náhodou nenajde. Testování tohoto scénáře probíhalo venku, kde jsem nechal zařízení položené v lese na pařezu, a zavolal kolegovi doma u PC, aby přes webovou aplikaci zařízení zamknul a lokalizoval. V případě, že bylo zařízení na mobilním signálu (s alespoň EDGE připojením k internetu) se chovalo vše korektně. Lokalizace nebyla úplně přesná, ale když člověk tuší kde se pohyboval, neměl by mít problém zařízení nalézt. Problém nastává, když zařízení na signálu není. Příkaz na něj nemůže dorazit a dorazí, až když zařízení někdo vezme a přenese na signál. V tu chvíli se zamkne a lokalizuje správně. Tento problém ovšem nelze vyřešit.

Testování pokračovalo scénářem *„Ztracený telefon našel ten, kdo ho chce vrátit“* rozepsaném v kapitole 5.1.3. V podstatě se jedné o podobný scénář jako minule, akorát zařízení někdo nalezl a je ochotný ho majiteli vrátit. Z tohoto důvodu je na uzamykací obrazovce tlačítko pro zavolání majiteli. Toto číslo a také zpráva, která se objeví na displeji, se zadává z webové aplikace při zamknutí zařízení. Přijímání příkazů i zasílání zpráv fungovalo dle předpokladů. Ovšem, objevil se problém s vyvoláním

[4] Device-independet pixel nebo také density-independent pixel – virtuální jednotka, která se přizpůsobuje podle hustoty pixelů daného displeje.

*intent*u pro vytočení telefonního čísla když je zařízení uzamknuté. Řešení bylo deaktivovat před zavoláním uzamknutí a po ukončení hovoru ho opět aktivovat. Tato oprava byla zahrnuta do finální verze. Další problém je samozřejmě s tabletem, ze kterého nejde telefonovat. Ovšem jde přečíst číslo pro volání, takže ten kdo chce zařízení vrátit, může zavolat zpět ze svého telefonu. Také je možné jako zprávu na displej zaslat například svou emailovou adresu.

Následovalo testování funkcí ohledně scénáře, že *telefon nalezl někdo, kdo nemá v úmyslu ho vrátit*, případně ho ukradl, popsaném v kapitole 5.1.4 spolu se scénářem, kdy předpokládáme, že se nám zařízení už nikdy nevrátí specifikovaném v kapitole 5.1.6. Prioritami je zde tedy uzamknutí zařízení, zjištění jeho polohy, případě získání výpisů volání a SMS. Pokud je zařízení uzamknuté zasílají se také zprávy o změně SIM karty a o úspěšných i neúspěšných pokusech o odemknutí (včetně fotografie z přední kamery). Dále také možnost vymazat data na SD kartě a uvést zařízení do továrního nastavení. Testování probíhalo opět ve spolupráci s kolegou, kdy první z nás seděl doma u webové aplikace a druhý s mobilním zařízením vyrazil ven do města. Aplikace se chovala korektně ve všech případech. Jen foto z přední kamery chodí do aplikace otočené o 90° v případě telefonu. U tabletu tento problém není. Bylo také otestováno, jestli zařízení zůstává v uzamčeném stavy i po vyjmutí baterie a opětovném zapnutí a i v tomto případě se chová korektně.

Posledním testovaným scénářem byl „*ukradený telefon z kapsy*“ z kapitoly 5.1.5. Zde má telefon zareagovat na příchozí SMS zprávu ve tvaru CMD STOLEN a telefon uzamknout v tzv. stolen modu, kdy ještě spustí zvuk říkající, že tento telefon byl ukraden. Dalším SMS příkazem CMD UNSTOLEN je tento mód zrušen ale telefon je potřeba stále odemknout heslem, které přišlo v odpovědní SMS. Tato funkce bohužel není dostupná na tabletu, protože ten nemůže přijímat SMS. Testování proběhlo pouze v soukromí, aby hlášení o krádeži nevyvolalo rozruch. Telefon se signálem na SMS příkazy reagoval správě, občas ovšem s drobným opožděním mezi odesláním SMS z jednoho telefonu a přijmutím v druhém, způsobeným operátorem.

Všechny nalezené chyby, závady a nedostatky byli do finální verze opraveny. Aplikace tedy plní správně všechny funkce, které byly ve specifikaci požadavků zadány a později implementovány.

7.3 Možnosti dalšího rozšíření

Po navržení, implementaci a otestování aplikace si dovolím navrhnout některá další rozšíření, která by aplikaci mohla vylepšit, ale nejsou už obsahem této práce. Jednou z věcí co by si jak webová tak mobilní aplikace zasloužila je profesionálnější grafický návrh a GUI. V této verzi je webová aplikace pouze grafický prototyp postavený nad CSS frameworkem Bootstrap 3. Z něj se dá vyházet pro budoucí implementaci a kódování grafického návrhu.

Dále by šlo dodělat několik funkcí do mobilní aplikace. Což samozřejmě obnáší přidat nové typy příkazů, zpráv a jejich zobrazování ve webové aplikaci. Jedním z možných rozšíření by mohlo být nahrávání zvuků, které by sloužilo k odposlouchávání nálezce telefonu. Dále pak místo fotografie

z přední kamery posílat několika sekundové video. Jistě by také bylo příhodné posílat aktuální screenshoty z telefonu.

Z implementačního hlediska by bylo do budoucna nejlepší přepsat celou webovou aplikaci do jazyku Java například na Spring frameworku. Výhodou by bylo, že by se dali používat stejné Entitní i jiné třídy pro mobilní i webovou aplikaci. Dále taky spuštěná webová Java aplikace dokáže nechat běžet některé procesy stále na pozadí, což je v mém případě ideální pro XMPP daemona a Websocket server.

Ovšem největší nevýhodou aplikace je, že po zformátování telefonu zmizí a tudíž už je telefon k nenalezení. Na tuto situaci ovšem neexistuje žádní řešení.

8 Závěr

V rámci této práce jsem se seznámil s postupy a principy vývoje aplikací pro mobilní platformu Android, s různými balíčky Android SDK a s vývojovým IDE, Android Studiem. Tyto informace považuji za velmi hodnotné pro svůj osobní rozvoj a pozici na trhu práce, zejména v dnešní době, kdy se tento operační systém rozšiřuje i do jiných oblastí než jsou jen mobilní telefony či tablety (například hodinky, auta, televize apod.).

Dalším tématem této práce bylo také prostudovat principy a implementaci služby Google Cloud Messaging. Ukázalo se, že tato služba je optimální pro zasílání zpráv na mobilní zařízení připojená k internetu a to jednak díky datovým úsporám a šetření baterie, ale i díky podpoře seskupování stejných typů zpráv v případě, že je mobilní zařízení off-line a tedy dojde k doručení pouze poslední aktuální zprávy. GCM se také ukázalo optimální pro zasílání zpráv ze zařízení na server, kdy také dokáže pozdržet zprávy v případě, že je off-line a odeslat je po opětovném připojení k síti.

Po porovnání a zhodnocení několika konkurenčních řešení jsem navrhnul způsoby, které uživateli ztraceného mobilního telefonu pomůžou jej nejít, získat zpět od nálezce či zablokovat, aby telefon nemohl případný nálezce zneužít. Byly vybrány dle mého rozhodnutí nejlepší a ty se objevili ve specifikaci požadavků na moji aplikaci.

Hlavním tématem bylo navržení aplikace pro mobilní zařízení a její webovou část. Byla navržena její vizuální podobna i návrh tříd a databáze. Toto všechno proběhlo v rámci semestrálního projektu v prvním semestru.

V druhém semestru jsem na základě předchozí specifikace mobilní i webovou aplikaci implementoval. Využil jsem jazyk Java s Android SDK na straně mobilní aplikace a jazyk PHP 5.4 s využitím Nette Frameworku, HOA liberaries a dalších externích knihoven na straně webové aplikace.

Během implementace mobilní aplikace byli využity třídy pro lokaci zařízení různými způsoby, pro administraci zařízení *DeviceAdminPolicy*, pro ovládání Bluetooth, pro obsluhu fotoaparátu, pro přijímání a odesílání SMS, pro ovládání uzamykací obrazovky *KeyguardManagaer*, pro ovládání

přehrávání zvuku a ovládání hlasitosti, pro práci s Google Cloud Messaging, pro zjišťování informací o telefonu a další standartní pro běh aplikace.

Při implementaci webové aplikace vznikla samostatná externí PHP knihovna pro práci s Google Cloud Messagingem, kterou jsem dal veřejně na GitHub a do katalogu PHP balíčků packagist.org. V době dokončení této diplomové práce už má dokonce několik stažení.

Literatura

[1] HASHIMI, Sayed Y. *Pro Android 2*. New York: Apress, c2010, xvi, 718 s. ISBN 978-1-4302-2659-8.

[2] MEIER, Reto. *Professional Android 2 application development*. Indianapolis: Wiley, 2010, xxxii, 543 s. ISBN 978-0-470-56552-0.

[3] HERODEK, Martin. Android: jednoduše. 2. aktualiz. vyd. Brno: Computer Press, 2014, 128 s. Naučte se za víkend (Computer Press). ISBN 978-80-251-4298-1.

[4] OPEN HANDSET ALLIANCE. *Alliance Overview* [online]. [cit. 2014-12-28]. Dostupné z: http://www.openhandsetalliance.com/oha_overview.html

[5] IDC, *Smartphone OS Market Share, Q3 2014* [online]. 2014 [cit. 2014-12-28]. Dostupné z: http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[6] ANDROID SOURCE. *The Android Source Code* [online]. 2013 [cit. 2014-12-28]. Dostupné z: http://source.android.com/source/index.html

[7] PERLÍK, Jan. *Jak se vyvíjel operační systém Android? Napříč jeho historií až do současnosti* [online]. 2013-11-13 [cit. 2014-12-28]. Dostupné z: http://www.androidmarket.cz/android/jak-se-vyvijel-operacni-system-android-napric-jeho-historii-az-do-soucasnosti-1-dil

[8] ANDROID DEVELOPERS. *Dashboards* [online]. 2013 [cit. 2014-12-28]. Dostupné z: http://developer.android.com/about/dashboards/index.html

[9] DSL.SK. *Google ukázal nový Android L, je dvakrát výkonnejší, má dlhšiu výdrž a vyzerá inak* [online]. 26.6.2014 [cit. 2014-12-28]. Dostupné z: http://dsl.sk/article.php?article=15760

[10] ANDROID DEVELOPERS. *Services* [online]. 2013 [cit. 2014-12-28]. Dostupné z: http://developer.android.com/guide/components/services.html

[11] ANDROID DEVELOPERS. *Google Cloud Messaging for Android* [online]. 2013 [cit. 2014-12-28]. Dostupné z: https://developer.android.com/google/gcm/index.html

[12] ANDROID, *Android - 5.0 Lollipop* [online]. 2014 [cit. 2015-01-10] Dostupné z: http://www.android.com/versions/lollipop-5-0/

[13] ANDROID DEVELOPERS. *Activities* [online]. 2013 [cit. 2015-01-10]. Dostupné z http://developer.android.com/guide/components/activities.html

[14] ANDROID DEVELOPERS. *Manifest* [online]. 2013 [cit. 2015-05-01]. Dostupné z http://developer.android.com/guide/topics/manifest/uses-sdk-element.html

[15] ANDROID, *Android - 4.3 Jelly Bean* [online]. 2013 [cit. 2015-05-01] Dostupné z: http://www.android.com/versions/jelly-bean-4-3/

[16] IKKINK, Hubert K. *Gradle effective implementation guide*. Birmingham, [Eng.]: Packt Publishing, 2012. ISBN 9781849518116.

[17] ANDROID DEVELOPERS. *Device Admin* [online]. 2013 [cit. 2015-03-15]. Dostupné z http://developer.android.com/guide/topics/admin/device-admin.html

[18] TICHÝ, Jan. *Doctrine 2: základní definice entit* [online]. 2010 [cit. 2015-03-15]. Dostupné z: http://www.zdrojak.cz/clanky/doctrine-2-zakladni-definice-entit/

[19] W3C. *Server-Sent Events: W3C Recommendation 03 February 2015* [online]. 2015 [cit. 2015-04-05] Dostupné z: http://www.w3.org/TR/eventsource/

[20] W3C, *The WebSocket API: W3C Candidate Recommendation 20 September 2012* [online]. 2012 [cit. 2015-04-05] Dostupné z: http://www.w3.org/TR/websockets/

[21] FETTE, I. *The WebSocket Protocol* [online]. 2011 [cit. 2015-04-25]. Dostupné z: http://tools.ietf.org/html/rfc6455

[22] HOA-PROJECT. *Hack book of Hoa\Websocket* [online]. 2014 [cit. 2015-04-25]. Dostupné z http://hoa-project.net/En/Literature/Hack/Websocket.html

[23] THEFTIE. *Features*. [online]. 2014 [cit. 2015-04-25]. Dostupné z http://www.theftie.net/feature

Seznam příloh

Příloha 1. CD se zdrojovými kódy